Progetto Eulero: Problema 14

3 marzo 2008 A cura di Marco Beri

Nel precedente articolo sul progetto Eulero abbiamo saltato dal terzo al quindicesimo problema. L'unico modo per non perdere l'abitudine alla nostra imprescindibile *aperiodicità* era quello di tornare indietro e infatti ci occuperemo questa volta del problema numero 14.

Leggiamo assieme l'enunciato:

> *La seguente sequenza iterativa è definita nell'insieme dei numeri interi:*
>
> *n → n/2 (se n è pari)*
>
> *n → 3n + 1 (se n è dispari)*
>
> *Usando la precedente regola e partendo da 13, possiamo calcolare la seguente sequenza:*
>
> *13 → 40 → 20 → 10 → 5 → 16 → 8 → 4 → 2 → 1*
>
> *Si può vedere che questa sequenza (che comincia con 13 e finisce con 1) contiene 10 termini. Sebbene non è ancora stato provato (congettura di Collatz), si ritiene che, qualunque sia il numero di partenza, si finisca sempre a 1.*
>
> *Partendo da quale numero, al di sotto di un milione, si ottiene la catena più lunga?*

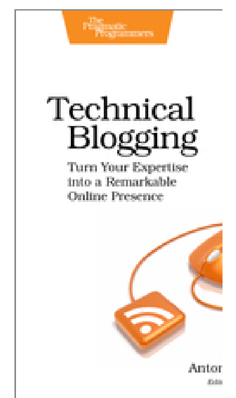
La congettura di Collatz è un problema molto curioso perchè, nella sua semplicità, sfugge dal 1937 alla dimostrazione della sua verità o meno. Il grandissimo Paul Erdös disse "la matematica non è ancora pronta per questo genere di problemi".

Nella figura seguente è mostrato il grafico del numero di passi necessari per arrivare a 1 partendo dai numeri fino a 9999.

Per fortuna noi non dobbiamo dimostrare la congettura ma solamente trovare il numero più *lento* sotto il milione.

SEGUITO DA PIÙ DI 16,000 PR

SCARICA IL MIO LIBRO



Ora disponil

404 Not F

nginx/1.19.

POST RECENTI

Recensione di Amazon B

Non ci sono grossi problemi nel tradurre l'enunciato in poche righe di codice **Python** e quando si risolvono i problemi del progetto Eulero è sempre buona norma provare il codice con l'esempio fornito nell'enunciato. Per questo motivo testiamo il nostro codice con il valore 13:

```
l = 1
next_n = 13
while next_n != 1:
    if next_n % 2 == 0:
        next_n = next_n / 2
    else:
        next_n = next_n * 3 + 1
    l += 1
print l
```

Eseguiamo il programma e vediamo cosa succede:

```
python 14.py
10
```

La seguenza generata dal valore iniziale 13 è effettivamente lunga 10 elementi come ci diceva l'enunciato stesso del problema, per cui possiamo generalizzare il codice per i valori fino a un milione:

```
max_l = 0
max_n = 0
for n in range(2, 1000000):
    l = 1
    next_n = n
    while next_n != 1:
        if next_n % 2 == 0:
            next_n = next_n / 2
        else:
            next_n = next_n * 3 + 1
        l += 1
    if l > max_l:
        max_l = l
        max_n = n
print max_n, max_l
```

Proviamo ad eseguire il codice:

```
python 14.py
Traceback (most recent call last):
  File "14.py", line 11, in <module>
    l += 1
KeyboardInterrupt
```

Cosa è successo? Semplice: dopo più di un minuto di attesa ci siamo stufati e abbiamo interrotto con un bel CTRL-C l'esecuzione del nostro programma.

Una delle regole del progetto stabilisce in un minuto il tempo massimo di esecuzione. Certo, avremmo potuto attendere un paio di minuti e probabilmente il nostro programma avrebbe terminato comunque, ma se già al problema 14 deroghiamo al tempo di esecuzione, cosa faremo al problema 184? Attenderemo un paio di anni?

Il vero nemico degli artis

10 consigli per interagire geek di famiglia

Amazon lancia il Kindle i

IBM rilascia la versione 9 Express-C

COMMENTI RECENTI

Leonardo su 10 consigli il proprio geek di famigli

Gilton su 10 consigli per proprio geek di famiglia

Lorenzo su La fluidità de

Lorenzo su 10 consigli pe proprio geek di famiglia

Mr.Price su 10 consigli p proprio geek di famiglia

CATEGORIE

Applicazioni & OS (13)

Editoriali (11)

Gadget (7)

IT Business (25)

Legge & Privacy (8)

Libri (6)

Networking & Security (1

Programmazione (104)

Siamo tutti geek (33)

Software Engineering (9)

Tlc & Internet (17)

TAG POPOLARI

agile algoritmi amazon ap conferenze django eve programming firefox framew internet ironruby italia java l linguaggi linux lisp m microsoft mozilla netw Programmazione project e python rails rest

Proviamo a ragionare sul nostro codice. Cosa possiamo fare per migliorare l'esecuzione? Scriviamo su un foglio le sequenze generate dai primi numeri:

$2 \to 1$

$3 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1$

$4 \to 2 \to 1$

$5 \to 16 \to 8 \to 4 \to 2 \to 1$

$6 \to 3 \to 10 \to 5 \to 16 \to 8 \to 4 \to 2 \to 1$

Spesso *guardare* con gli occhi i dati di un problema può fornirci il suggerimento vincente. In questo caso possiamo accorgerci che quando calcoliamo la sequenza del numero 6, arriviamo al 3 e da quel punto in poi ripetiamo pedissequamente i suoi stessi passaggi.

Proviamo a riscrivere il nostro programma in modo che il ciclo di calcolo si interrompa ogni volta che incontra un numero già calcolato in precedenza:

```
max_l = 0
max_n = 0
cache = {1:1}
for n in range(2, 1000000):
    l = 0
    next_n = n
    while next_n >= n:
        if next_n % 2 == 0:
            next_n = next_n / 2
        else:
            next_n = next_n * 3 + 1
        l += 1
    l = l + cache[next_n]
    cache[n] = l
    if l > max_l:
        max_l = l
        max_n = n
print max_n
```

Ora l'esecuzione si conclude in circa 5 secondi. Niente male per un linguaggio dinamico interpretato come Python.

Questa volta non forniamo la soluzione del problema, però vi diciamo che la sequenza più lunga ottenibile con un numero di partenza inferiore al milione è composta da ben 525 elementi. Inoltre il numero di operazioni che avremmo dovuto calcolare senza usare il nostro *escamotage* è altissimo, superiore a 100 milioni (di nuovo per il numero preciso dovrete impegnarvi un pochino da soli 🙂).

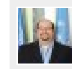
Post simili:

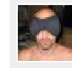
1. Progetto Eulero: Problema 1
2. Progetto Eulero: Problema 15
3. Progetto Eulero: Problema 2
4. Progetto Eulero: Problema 3

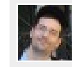
Filed Under: Programmazione   Tagged With: paul-erdos, progetto-eulero, python

scheme startup vig
web web2.0 windows xul

HANNO COLLABORATO

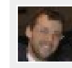
 Antonio Cangianc

 Marco Beri (RSS) (

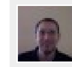
 Michele Simionat

Alex Beri (RSS) (1

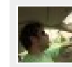
Ludovico Magnoc

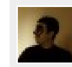
 Marco Ceresa (RS

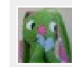
Valentino Volongh

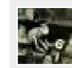
 Davide Ficano (RS

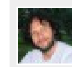
Piergiuliano Boss

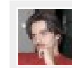
 Simone Dall'Ange

 Giovanni Intini (R

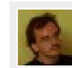
 Gabriele Renzi (R

 Nicholas Wieland

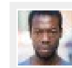
 Daniele Varrazzo

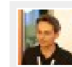
 Luigi Panzeri (RSS

Michele Finotto (

 Stefano Rodighie

Matteo Nodari (R

 Lawrence Oluyed

 Andrea Righi (RSS

About Marco Beri

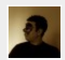
Marco Beri si laurea in Scienze dell'Informazione nel 1990, periodo oramai definibile come la preistoria del settore. Il computer è prima di tutto un suo hobby e anche per questo si innamora di Python a prima vista nel lontano 1999, dopo aver sperimentato una ventina di altri linguaggi. Fa di tutto, riuscendoci, per portarlo nella sua azienda, la Link I.T. spa, dove dal 1997 occupa il ruolo di amministratore e responsabile dello sviluppo software. Riesce perfino a intrufolarsi come amministratore nella fondazione dell'associazione Python Italia. Incredibilmente pubblica anche diversi libri per Apogeo/Feltrinelli.

Comments

maelstrom says:
3 marzo 2008 at 10:12

L'uso di cache per riutilizzare sotto-soluzioni già calcolate è un must nella programmazione dinamica e può, in generale, migliorare di molto l'efficienza temporale di un algoritmo, sebbene potrebbe inficiare l'efficienza spaziale (ma non è questo il caso!).

Inoltre sono particolarmente d'accordo sul fatto di visualizzare i dati del problema, spesso forniscono un'inaspettata chiave di lettura!



Giovanni Intini says:
3 marzo 2008 at 10:17

Articolo meraviglioso, questa serie si conferma una delle gemme di StackTrace 🙂

Antonio Cangiano says:
3 marzo 2008 at 10:44

Ho scoperto una prova veramente rimarchevole di ciò, ma questo margine è troppo piccolo per contenerla. 😛

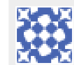
Ottimo articolo Marco. 🙂

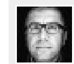
Michele Simionato says:
3 marzo 2008 at 20:52

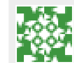
E' tradizione dare le soluzioni dei problemi del progetto Eulero in diversi linguaggi, compresi quelli piu' esotici. Ti mando quindi due soluzioni in R6RS Scheme:

1. una soluzione completamente funzionale, che non usa l'assegnazione:

```
(define (transform n)
(if (even? n) (/ n 2) (+ (* 3 n) 1)))

(define (add-sequence-length n cache)
(let loop ((i 0) (n* n))
(let ((slot* (assq n* cache)))
(if slot*
(append cache `((,n . ,(+ i (cdr slot*)))))
(loop (+ 1 i) (transform n*))))))

(define (get-max pairs)
(fold-left
(lambda (a p)
```


Carmine Noviello

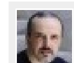

Claudio Cicali (RS


Alberto Revelli (R

Roberto De Ioris (

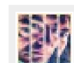
Davide Casali (RS

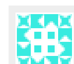
Franco Lombardo

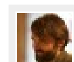

Nicola Larosa (RS

Lorenzo Bolognin

Massimiliano Mir


Emmanuele Som


Francesco Matara


Marco De Paoli (R

Massimo Scamar

(let ((n (car p)) (l (cdr p)) (n* (car a)) (l* (cdr a)))
(if (> l l*) (cons n l) (cons n* l*)))) '(1 . 1) pairs))

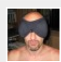
(define (longest-sequence N)
(let loop ((n 2) (cache '((1 . 1))))
(if (> n N) (get-max cache)
(loop (+ n 1) (add-sequence-length n cache)))))

(longest-sequence 10000)

Questa soluzione e' lenta perche' fa delle ricerche
O(N) su liste associative molto lunghe.

2. Una soluzione veloce e' usare dizionari
come in Python, pero' non e' funzionale:

```
(define (transform n)
  (if (even? n) (/ n 2) (+ (* 3 n) 1)))

(define cache (make-eq-hashtable))
(hashtable-set! cache 1 1)

(define (add-sequence-length n)
  (let loop ((i 0) (n* n))
    (let ((c (hashtable-ref cache n* #f)))
      (if c
          (hashtable-set! cache n (+ i c))
          (loop (+ 1 i) (transform n*))))))

(define (get-max)
  (define n 1)
  (define l 1)
  (vector-for-each
   (lambda (n*)
     (define l* (hashtable-ref cache n* #f))
     (when (> l* l) (set! l l*) (set! n n*)))
   (hashtable-keys cache))
  (cons n l))

(define (longest-sequence N)
  (let loop ((n 2))
    (if (> n N) (get-max)
        (begin (add-sequence-length n) (loop (+ n 1))))))

(longest-sequence 1000000)
```

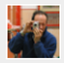


marcob says:
3 marzo 2008 at 21:28

@Michele: grazie!
A questo punto rispondo col il codice assembly di bitRake, uno degli iscritti al Progetto Eulero

```
mov edx, 1000000
   xor ebp, ebp ; max chain
   xor ebx, ebx

next:   dec edx
   je done
   cmp ebp, ebx
   mov eax, edx
   cmovc ebp, ebx ; save chain length
```

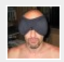
```
    cmovc edi, edx ; save number
    xor ebx, ebx
    jmp _even
_odd:
    lea eax, [eax][eax*2][2]
    inc ebx
_even:
    inc ebx
    shr eax, 1
    je next
    jc _odd
    jmp _even
done:
    inc edi ; answer
```



professore says:
16 marzo 2008 at 14:34

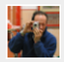
Vorrei dare un consiglio: potreste annunciare quale sarà il prossimo problema del progetto Eulero che tratterete, così possiamo provare a pensarci prima che voi pubblichiate l'articolo.

Ciao e complimenti!



marcob says:
16 marzo 2008 at 15:05

@professore: perché non ne proponi uno tu? Non è sempre facile trovare un problema scelto tra i primi (non vogliamo favorire troppo i lettori di Stacktrace 😉) che dia spunto per qualche riflessione.

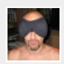


professore says:
16 marzo 2008 at 16:56

Ah, questo significa che mi devo mettere *davvero* a risolverli prima che voi lo facciate 🙂

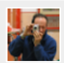
Tra i primi vedo il problema 6, ad esempio. Anche se richiama un po' il primo, ed è risolvibile senza l'uso di un programma.

Oppure il 16, per il quale non "vedo" una soluzione senza usare programmi (bisogna che ci pensi un po' (ok, vada per questo, mi attira di più dell'altro perché devo pensarci sopra…)).



marcob says:
16 marzo 2008 at 17:04

@professore: per risolvere il 16, stampa compresa, sono sufficienti 33 caratteri in Python. Vediamo se tu o qualcun altro sa trovarli o anche fare di meglio 🙂



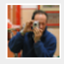
professore says:
16 marzo 2008 at 22:14

Mh, io cercavo una soluzione con carta e penna, ma vedo che con un programma si fa molto in fretta.

Essendo io novizio di python, non sono sceso a 33 caratteri, ne ho un po' di più. Ecco la mia soluzione

```
n=str(2**1000)
s=0
for i in n:
  s=s+int(i)
print s
```

Ciao

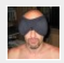


professore says:
16 marzo 2008 at 22:14

(Ehm, si è persa l'indentazione)

C8E says:
16 marzo 2008 at 22:19

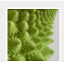
Ritrovata 😉



marcob says:
16 marzo 2008 at 22:39

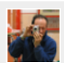
@professore: Bravo! Comunque si può fare in 32 (avevo lasciato uno spazio di troppo).

Anzi, considerato che 2**1000 == 4**500, si può fare in 31… 🙂



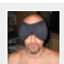
ric says:
16 marzo 2008 at 23:16

e se non prendo un abbaglio, anche in 25..



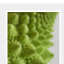
professore says:
16 marzo 2008 at 23:16

Per farlo in 32 mi sa che devo andare avanti nella lettura del tutorial di python, si useranno cose ad alto livello che ancora non conosco… [come si fa per mantenere l'indentazione del codice?]



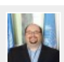
marcob says:
16 marzo 2008 at 23:22

@ric: senza il print si può fare in 25. Ma con il print io arrivo a 31.



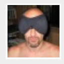
ric says:
16 marzo 2008 at 23:29

@marcob: gia`, pure io.. 🙂 scusate la scemata, mi pareva strano in effetti.



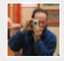
Antonio Cangiano says:
16 marzo 2008 at 23:31

@professore, usa il tag pre.



marcob says:
16 marzo 2008 at 23:43

@professore: si usano (oltre al print) esattamente 4 funzioni di 3 caratteri. E sono tutte nel tutorial… 🙂

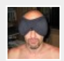


professore says:
22 marzo 2008 at 12:14

Rieccomi. Ho provato a risolvere alcuni problemi del progetto, e poi mi sono dedicato a quello uscito stamattina, il numero 187; nel frattempo mi sono letto un po' di tutorial di python.

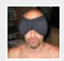
Per risolvere il problema 187 serve un generatore di numeri primi (a meno di non barare cercando con google il valore della funzione pi(x), funzione che conta i numeri primi minori di x). Voi ne avete già proposto uno, ma in python è troppo lento per gli scopi del problema in questione. Io me ne ero scritto uno in C, tempo fa, che non usa però le tecniche usate da voi, ma semplicemente controlla se un numero può essere fattorizzato mediante i numeri primi già generati in precedenza. Troppo lento anche quello, su una vecchia macchina Pentium 4 a 1600Mhz.

Quindi la mia proposta per il prossimo problema è questa: uno qualsiasi tra quelli che richiedono un generatore di numeri primi molto veloce.



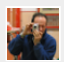
marcob says:
22 marzo 2008 at 13:21

@professore: il 187 è facilmente risolubile in Python se si è già salvata una lista di numeri primi fino a (10**8)/2. Oggi proverò qualche algoritmo veloce in Python per generare questa lista per vedere se si riesce a rispettare il limite del minuto.



marcob says:
22 marzo 2008 at 14:07

@professore: ringraziando il sig. Atkin e il sig. Bernstein, si possono generare in Python i numeri primi tra 1 e (10**8)/2 in circa 28 secondi.



professore says:
22 marzo 2008 at 14:15

Vado a cercare i riferimenti, grazie.

Policy per i commenti: Apprezzo moltissimo i vostri commenti, critiche incluse. Per evitare spam e troll, e far rimanere il discorso civile, i commenti sono moderati e prontamente approvati poco dopo il loro invio.

COPYRIGHT © 2007-201