The Wayback Machine - https://web.archive.org/web/20170226154307/http://stacktrace.it/2009/05/20/una-scatola-degli-attrezzi-euleriana/

Una scatola degli attrezzi Euleriana

20 maggio 2009 A cura di Marco Beri

Nell'archivio di Stacktrace ci sono diversi articoli che si sono occupati del Progetto Eulero. Invece di risolvere un particolare problema, questa volta daremo uno sguardo *pythonista* ad alcuni strumenti che possono servirci nella risoluzione delle sfide del Progetto.

Perché Python? Beh, a quanto pare Python è il linguaggio preferito dagli italiani che si dilettano con i problemi di Eulero, non per nulla dei primi 10 della classifica tricolore, ben 6 lo usano!

	Username	Country	Level	Solved	Current Performance	Language
1	sfabriz		5	245		Python
2	Taifu		5	245		Python
3	fra		5	244		Java
4	Giulio		5	240		Python
5	burhomer		4	197		Java
6	mastro		4	180		Python
7	Scidomino		4	166		Java
8	lanidrak		4	163		APL/J/K
9	adnapli		4	160		Python
10	srskko		3	146		Python

Il decoratore Cronometro

Una delle regole più importanti del Progetto Eulero è la cosiddetta "one-minute-rule": tutti i problemi sono pensati in modo da non richiedere tempi di esecuzione superiori al minuto su un normale personal computer di qualche anno fa.

Quale occasione migliore per un decoratore che misuri il tempo di esecuzione di una funzione?

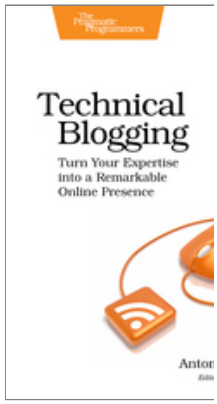
```
# File cronometro.py
import time
def cronometro(funzione):
    def funzione_decorata(*args, **kwargs):
        inizio = time.time()
        valore_ritorno = funzione(*args, **kwargs)
        print 'Esecuzione di %s in %.2f secondi' % (
            funzione.func_name, time.time() - inizio)
        return valore_ritorno
    return funzione_decorata
```

Proviamo il nostro decoratore con una semplice funzione:

SEGUITO DA PIÙ DI 16,000 PR

Hai **idee** per un arti

SCARICA IL MIO LIBRO



Ora disponil

Follow @stacktrace

404 Not F

nginx/1.19.

POST RECENTI

Recensione di Amazon B

```
# File test1.py
from cronometro import cronometro

@cronometro
def potenza_2(n):
    cont = 0
    while cont < 2**n:
        cont += 1
    return n, cont

for i in (10, 15, 20, 25):
    print potenza_2(i)
```

L'esecuzione del precedente codice *decorato* da il seguente risultato:

```
$ python test1.py
Esecuzione di potenza_2 in 0.00 secondi
(10, 1024)
Esecuzione di potenza_2 in 0.02 secondi
(15, 32768)
Esecuzione di potenza_2 in 0.55 secondi
(20, 1048576)
Esecuzione di potenza_2 in 19.42 secondi
(25, 33554432)
```

Incidentalmente, con questo semplicissimo esempio, abbiamo modo di sperimentare una situazione molto comune con i problemi di Eulero: le prime versioni delle funzioni, spesso scritte con la tecnica della forza bruta, funzionano egregiamente per piccoli valori delle variabili dei problemi, mentre per quelli reali, se non cambiamo gli algoritmi, possiamo attendere con pazienza la fine dell'universo prima di avere la risposta.

Per i valori 10, 15 e 20 il tempo di esecuzione è sotto il secondo, con 25 saliamo a 19 secondi, con 30 l'algoritmo impiegherebbe 10 minuti a terminare e con 35 ci vorrebbero quasi 6 ore.

Il decoratore Memoize

A proposito di cambio di strategia, la *memoizzazione* è una potente tecnica della programmazione dinamica e spesso da sola è in grado di togliere le nostre castagne dal fuoco.

In cosa consiste? Semplice: si tratta di salvare il risultato derivato dall'esecuzione di una funzione, abbinandolo ai parametri con cui la funzione è stata chiamata, per poterlo eventualmente riusare istantaneamente in un secondo momento, nel caso la funzione sia chiamata con gli stessi valori.

Una versione molto semplice di memoize potrebbe essere la seguente:

```
# File memoize.py

def memoize(funzione):
    cache = {}
    def funzione_decorata(*args):
        try:
            return cache[args]
        except KeyError:
            ret = funzione(*args)
            cache[args] = ret
            return ret
    return funzione_decorata
```


Il vero nemico degli artis

10 consigli per interagire geek di famiglia

Amazon lancia il Kindle it

IBM rilascia la versione 9 Express-C

COMMENTI RECENTI

Leonardo su 10 consigli p 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)

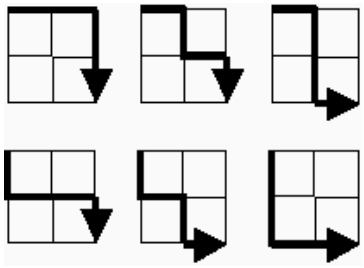
Software Engineering (9)

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

Vediamo un esempio pratico proprio con il quindicesimo problema di Eulero che abbiamo già analizzato in un precedente articolo su Stacktrace:

> *Partendo dall'angolo in alto a sinistra in una griglia di dimensione 2 per 2, ci sono 6 percorsi che vanno (senza tornare indietro) fino all'angolo in basso a destra:*
>
> 
>
> *Quanti percorsi simili ci sono in una griglia di dimensione 20 per 20?*

Proviamo ad attaccare il problema con una funzione ricorsiva che *esplora* in lungo e in largo la griglia.

Notiamo innanzitutto che quando arriviamo sul bordo inferiore o su quello destro della griglia abbiamo un solo modo di arrivare all'angolo in basso a destra.

Scriviamo quindi la nostra funzione esploratrice:

```
# File p15.py
from cronometro import cronometro

@cronometro
def eulero15(riga, colonna):
    def esplora(riga, colonna):
        # Bordo destro
        if colonna == 0:
            return 1
        # Bordo inferiore
        if riga == 0:
            return 1
        # Altrimenti esploro scendendo di una riga
        # e poi andando a destra di una colonna
        return esplora(riga - 1, colonna) + esplora(riga, colonna - 1)

    return esplora(riga, colonna)

if __name__ == "__main__":
    import sys
    print eulero15(int(sys.argv[1]), int(sys.argv[2]))
```

Proviamo ad eseguire il programma con una griglia di 2 per 2 di cui conosciamo già il numero di strade (6, ce lo dice il testo problema):

```
$ python p15.py 2 2
Esecuzione di eulero15 in 0.00 secondi
6
```

Ottimo!

Proviamo con qualche griglia più grossa:

scheme startup vig
web web2.0 windows xul

HANNO COLLABORATO

Antonio Cangianc
Marco Beri (RSS) (
Michele Simionat
Alex Beri (RSS) (18
Ludovico Magnoc
Marco Ceresa (RS
Valentino Volongh
Davide Ficano (RS
Piergiuliano Boss
Simone Dall'Ange
Giovanni Intini (R
Gabriele Renzi (RS
Nicholas Wieland
Daniele Varrazzo
Luigi Panzeri (RSS
Michele Finotto (F
Stefano Rodighier
Matteo Nodari (RS
Lawrence Oluyed
Andrea Righi (RSS

```
$ python p15.py 10 10
Esecuzione di eulero15 in 0.25 secondi
184756
$ python p15.py 15 15
Esecuzione di eulero15 in 254.34 secondi
155117520
$ python p15.py 20 20
Traceback (most recent call last):
...
  File "15.py", line 14, in esplora
    return esplora(riga - 1, colonna) + esplora(riga, colonna - 1)
KeyboardInterrupt
```

Purtroppo con la griglia di 20 per 20 abbiamo dovuto interrompere l'esecuzione oppure questo articolo non sarebbe mai uscito!

Per essere precisi sarebbe comunque uscito, ma in forte ritardo: 62 ore di attesa sono davvero troppe.

Non ci resta che provare a *memoizzare* la nostra funzione:

```
# File p15.py
from cronometro import cronometro
from memoize import memoize

@cronometro
def eulero15(riga, colonna):
    @memoize
    def esplora(riga, colonna):
        # Bordo destro
        if colonna == 0:
            return 1
        # Bordo inferiore
        if riga == 0:
            return 1
        # Altrimenti esploro in scendendo e andando a destra
        return esplora(riga - 1, colonna) + esplora(riga, colonna - 1)

    return esplora(riga, colonna)

if __name__ == "__main__":
    import sys
    print eulero15(int(sys.argv[1]), int(sys.argv[2]))
```

Riproviamo con la griglia 20 per 20:

```
$ python p15.py 20 20
Esecuzione di eulero15 in 0.00 secondi
137846528820
```

Zero secondi… difficile fare di meglio. E con due sole righe di codice in più! 🙂

Questa versione di memoize è semplicissima e non tiene conto di diversi potenziali problemi.

Per esempio se gli argomenti della funzione memoizzata non sono *hashabili* (usabili come chiavi di un dizionario), verrà sollevata un'eccezione:

 Carmine Noviello
 Claudio Cicali (RS
 Alberto Revelli (R
 Roberto De Ioris (
 Davide Casali (RS
 Franco Lombardo
 Nicola Larosa (RS
 Lorenzo Bolognin
 Massimiliano Mir
 Emmanuele Somı
 Francesco Matara
 Marco De Paoli (R
 Massimo Scamar

```
# File er.py
from memoize import memoize

@memoize
def pippo(a):
    return a

pippo([])
```

Un oggetto *list* è mutabile (i suoi elementi possono essere aggiunti, cambiati o cancellati) per cui non ci è permesso usarlo come chiave di un dizionario. Uno dei principali usi delle *tuple* (una versione immutabile delle liste) è proprio questo.

Comunque verifichiamo cosa succede eseguendo il precedente programma:

```
$ python er.py
Traceback (most recent call last):
  File "er.py", line 7, in
    pippo([])
  File "memoize.py", line 7, in funzione_decorata
    return cache[args]
TypeError: list objects are unhashable
```

Come volevasi dimostrare…

Per ovviare al problema possiamo usare il modulo *pickle* che trasforma ogni oggetto in una stringa (questa **ricetta** usa questo approccio).

Un'altra possibile insidia può derivare dal consumo eccessivo di memoria, in caso di funzione chiamata troppe volte con argomenti differenti. In questo caso ci sono altre soluzioni, come ad esempio **cancellare** i valori non usati da più tempo.

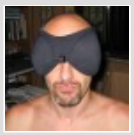
Ad ogni modo, a proposito dei problemi di Eulero, possiamo accontentarci della nostra versione semplificata del decoratore Memoize.

Conclusione

Questi sono un paio tra gli *strumenti* più semplici del bravo euleriano pythonista. Ce ne sono molti altri più avanzati, a partire dai generatori di numeri primi (un esempio è già stato affrontato da Daniele Varrazzo nel suo bellissimo **articolo** sul problema numero 3), passando dagli algoritmi per testare probabilisticamente la primalità e arrivando a quelli per fattorizzare numeri molto grossi o alla fantomatica funzione **Phi di Eulero**.

Vi interessano? 🙂

Filed Under: Siamo tutti geek Tagged With: matematica, progetto-eulero, python



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

foobar says:
24 maggio 2009 at 05:29

Yeah, a @memo decorator really helps (a lot). Cool site (though i cant read Italian..)

Hmm, and any hints for problem 246?

zar says:
24 maggio 2009 at 09:50

Sì! 🙂

marcob says:
24 maggio 2009 at 23:32

@foobar: ahah! Just solved (I'm Taifu). How much for the answer? 😀

foobar says:
25 maggio 2009 at 02:50

Hi, marcob (Taifu)

Congrats!

I'm not usually good at geometry :-(.. Totally no idea for this one. Not the answer, man, maybe some pointer to the needed math background? How about a bottle of beer for that? 😛 Is the problem end up being solved very elegantly by some beautiful math? Or it's just another some kind of (smart) brute?

foobar says:
25 maggio 2009 at 10:07

never mind.

just solved that. =) nice problem.

marcob says:
25 maggio 2009 at 11:17

@foobar: what's your login name on projecteuler? There are both approaches in the forum but a couple of things are requested to exploit them: to know the ellipse equation and to be able to find when a line is tangent to a curve (In how many points does a tangent intersect a curve? Zero? One? Two? More?)

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