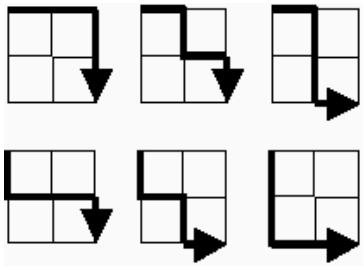
Progetto Eulero: Problema 15

4 February 2008 A cura di Marco Beri

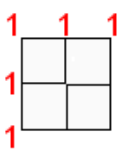
?

Dopo aver affrontato in sequenza i primi tre problemi del progetto Eulero, per mantener fede alla nostra onorata *aperiodicità* saltiamo a piè pari al problema numero 15.

Con questo problema abbandoniamo, almeno in apparenza, numeri e sequenze e per la prima volta affrontiamo un enunciato che richiede un'immagine:

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

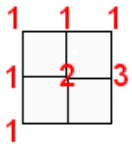
Per risolvere questo problema dobbiamo usare un po' di spirito di osservazione. Cominciamo a vedere che tutti i punti della griglia sul bordo superiore e sul bordo sinistro possono essere raggiunti in un unico modo:



Questo è abbastanza scontato visto che nell'enunciato è espressamente dichiarato che non possiamo *tornare indietro* nel nostro cammino che ci deve portare al bordo inferiore destro della griglia. Proviamo ora a concentrarci sul punto indicato dal cerchio blu:

In quanti modi è possibile raggiungerlo? Possiamo arrivarci dal punto superiore oppure dal punto alla sua sinistra. Quindi possiamo arrivarci con 2 percorsi:

Prendiamo ora in considerazione il punto subito a destra: possiamo arrivarci spostandoci a destra dal punto centrale (come abbiamo visto a sua volta raggiungibile in 2 modi) oppure scendendo dall'angolo in alto a destra (raggiungibile in un unico modo). In totale possiamo usare 3 percorsi distinti:



Oramai dobbiamo avere capito qual è la soluzione: ogni punto può essere raggiunto in tanti modi quanto è la somma dei percorsi del punto superiore e del punto a sinistra. Proviamo a compilare tutta la griglia e verifichiamo che nell'angolo in basso a destra otteniamo il 6 che è proprio il numero totale di percorsi indicati nell'enunciato:

Adesso che abbiamo scoperto come calcolare la soluzione, proviamo a scrivere un semplice programma in Python per trovare il numero desiderato:

```
def problem15(k):
    g = [[1 if i==0 or j==0 else 0 for i in range(k+1)]
                                   for j in range(k+1)]
    for y in range(1, k+1):
        for x in range(1, k+1):
                g[y][x] = g[y-1][x] + g[y][x-1]
    return g[k][k]
```

Mettiamo ora alla prova il nostro programma con la griglia 2 per 2 dell'enunciato:

```
>>> print problem15(2)
6
```

Confortati dal risultato corretto, proviamo adesso con la griglia di 20 per 20:

```
>>> print problem15(20)
137846528820
```

Il risultato è quello giusto!

Ora, come all'ingresso della griglia del problema, ci troviamo di fronte a due strade: accontentarci della soluzione trovata o provare a fare di meglio.

A qualcuno di voi forse sarà già suonato un campanello in testa osservando la griglia di 2 per 2 con i numeri ad ogni incrocio:

Proviamo a girare di 45 gradi la griglia:

Ancora nessun campanello? Questo è davvero l'ultimo su-su-suggerimento-to-to: Tartaglia!

Il triangolo di Tartaglia (o di Pascal per i francesi, di Khayyam per i persiani, di Stiefel per i tedeschi o di Zhu Shijie per i cinesi, ma per la cronaca sono arrivati primi i persiani) è una *piramide* di numeri facilissima da ottenere, sommando in ogni posizione i due numeri della riga precedente:

10 consigli per interagire geek di famiglia

Amazon lancia il Kindle i

IBM rilascia la versione 9 Express-C

Considerazioni sulla scar Italia

COMMENTI RECENTI

Gilton on 10 consigli per proprio geek di famiglia

Lorenzo on La fluidità de

Lorenzo on 10 consigli pe proprio geek di famiglia

Mr.Price on 10 consigli p il proprio geek di famigli

Havelock Vetinari on Il ve artisti in rete

CATEGORIE

Applicazioni & OS (13)

Editoriali (11)

Gadget (7)

IT Business (25)

Legge & Privacy (8)

Libri (6)

Networking & Security (1

Programmazione (104)

Siamo tutti geek (32)

Software Engineering (9)

Tlc & Internet (17)

TAG POPOLARI

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

```
                    1
                  1   1
                1   2   1
              1   3   3   1
            1   4   6   4   1
          1   5   10  10  5   1
        1   6   15  20  15  6   1
      1   7   21  35  35  21  7   1
```

A questo punto tutti dovremmo aver notato che i numeri in grassetto sono esattamente i numeri della nostra griglia girata di 45 gradi.

Gli elementi del triangolo di Tartaglia possono essere calcolati istantaneamente con la formula di Newton. In particolare, l'elemento *k-1 esimo* della riga *n-1 esima* è uguale a:

$$C_{n,k} = \binom{n}{k} = \frac{n!}{k! \cdot (n-k)!}$$

Proviamo quindi a scrivere un programma in Python che utilizzi questa formula:

```
def factorial(n):
    if n < 2:
        return 1
    return n * factorial(n - 1)

def newton(n, k):
    return factorial(n) / (factorial(k) * factorial(n - k))
```

Come abbiamo visto, il 6 della griglia 2 per 2 è rappresentato dal terzo elemento della quinta riga, quindi proviamo a eseguire la funzione *newton* passando i valori 4 e 2:

```
>>> print newton(4, 2)
6
```

Proviamo adesso con 40 e 20:

```
>>> print newton(40, 20)
137846528820
```

Ma quanto abbiamo risparmiato con questo nuovo approccio? Proviamo a misurarlo:

```
import timeit
t1 = timeit.Timer("problem15(20)", "from __main__ import problem15")
t2 = timeit.Timer("newton(40, 20)", "from __main__ import newton")
print "Problem15 % 4.2f" % (t1.timeit(10000))
print "Newton % 4.2f" % (t2.timeit(10000))
```

Le misurazioni sono abbastanza significative:

```
Problem15  4.00
Newton  0.53
```

Qualcuno potrà obiettare che l'uso di una funzione ricorsiva come *factorial* può andare incontro a problemi inaspettati con numeri più grandi e infatti:

```
>>> print newton(1000, 500)
```

```
Traceback (most recent call last):
  File "<pyshell#55>", line 1, in
    print newton1(1000, 500)
  File "<pyshell#53>", line 2, in newton1
    return factorial(n) / (factorial(k) * factorial(n - k))
  File "<pyshell#48>", line 4, in factorial
    return n * factorial(n - 1)
...
  File "<pyshell#48>", line 4, in factorial
    return n * factorial(n - 1)
RuntimeError: maximum recursion depth exceeded
```

Proviamo allora con una versione più *furba* della funzione Newton:

```
def newton_smart(n, k):
    newt = 1
    for i in range(n, n - k, -1):
        newt *= i
    for i in range(1, k + 1):
        newt /= i
    return newt
```

Mettiamola alla prova:

```
>>> print newton_smart(40, 20)
137846528820
>>> print newton_smart(1000, 500)
270288240945436569515614693625975275496152008446548287007392875106625428700
```

Misuriamo la sua velocità:

```
t3 = timeit.Timer("newton_smart(40, 20)",
                  "from __main__ import newton_smart")
print "Newton_smart %4.2f" % (t3.timeit(10000))
```

```
Newton_smart  0.21
```

Ancora meglio! Abbiamo migliorato di quasi un fattore 20 i tempi di esecuzione della prima versione del nostro programma.

In realtà possiamo ottimizzare ancora di più la nostra funzione. Dal punto di vista matematico calcolare il valore del binomio di Newton passando per esempio i valori *100, 2* piuttosto che *100, 98* è lo stesso:

```
>>> newton_smart(100, 98)
4950L
>>> newton_smart(100, 2)
4950
```

Il risultato è davvero lo stesso ma possiamo notare la piccola differenza della *L* che sta per *long* nel caso *100, 98*: questo è dovuto al fatto che la nostra funzione effettua ben 196 moltiplicazioni mentre nel caso *100, 2* ne effettua solo 4. Se proviamo a misurare i tempi di esecuzione vediamo una notevolissima differenza:

```
>>> t4 = timeit.Timer("newton_smart(100, 98)",
                      "from __main__ import newton_smart")
```

```
>>> t5 = timeit.Timer("newton_smart(100, 2)",
                      "from __main__ import newton_smart")
>>> print "Newton Smart 100-98 % 4.2f" % (t4.timeit(10000))
Newton Smart 100-98  1.41
>>> print "Newton Smart 100-2 % 4.2f" % (t5.timeit(10000))
Newton Smart 100-2  0.04
```

Possiamo allora scrivere una versione ancora più intelligente della funzione:

```
def newton_smartest(n, k):
    newt = 1
    if k > n/2:
        k = n - k
    for i in range(n, n - k, -1):
        newt *= i
    for i in range(1, k + 1):
        newt /= i
    return newt
```

```
>>> t6 = timeit.Timer("newton_smartest(100, 98)",
                      "from __main__ import newton_smartest")
>>> t7 = timeit.Timer("newton_smartest(100, 2)",
                      "from __main__ import newton_smartest")
>>> print "Newton Smartest 100-98 % 4.2f" % (t6.timeit(10000))
Newton Smartest 100-98  0.04
>>> print "Newton Smartest 100-2 % 4.2f" % (t7.timeit(10000))
Newton Smartest 100-2  0.04
```

Adesso, per dormire definitivamente sonni tranquilli, non ci resterebbe che controllare la correttezza dei valori passati alla nostra funzione (k dovrebbe essere non negativo e inferiore o uguale a n), ma questo possiamo lasciarlo come esercizio per i più volonterosi.

Per accontentare i più esoterici, invece, ecco una versione in Lisp che sfrutta la stessa formula:

```
 (defun ! (n)
  (apply #'* (loop for i from n downto 1 collect i)))

(defun number-of-paths (n m)
  (/ (! (+ n m)) (* (! n) (! m))))

(defun p15 ()
  (number-of-paths 20 20))
```

E ora qualcosa di completamente diverso:

Proprio così: è un foglio di calcolo che risolve il problema numero 15! È sufficiente inserire il valore 1 nella cella B1, la formula *=A1+B1* nella cella B2 e quindi copiarla fino alla cella V41 dove apparirà il risultato.

Infine, se non abbiamo a disposizione nemmeno un foglio di calcolo, proviamo con Google.

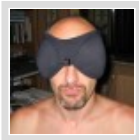
?

Post simili:

1. Progetto Eulero: Problema 1
2. Progetto Eulero: Problema 2

3. Progetto Eulero: Problema 3

Filed Under: Programmazione Tagged With: algoritmi, matematica, project euler, triangolo-di-tartaglia



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:
4 February 2008 at 21:30

Il triangolo di Tartaglia è utilissimo, specialmente per le sue evidenti relazioni con i coefficienti binomiali (o di Newton), ma sinceramente non l'avevo mai visto applicato a questo problema.

Tutti questi articoli sul progetto Eulero, bellissimi peraltro, non fanno che confermare la mia opinione dell'estrema importanza che riveste una buona formazione matematica sulla formazione di un informatico.

Gabriele says:
6 February 2008 at 12:27

Come ribadisco ad ogni nuovo problema svolto..continuate così, è veramente un piacere leggervi.

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.