Programmazione competitiva con PyPy: "Vincere con Python!"

Alessandro Amici <a.amici@bopen.it>

B-Open Solutions – http://bopen.it

Python già vince le competizioni!

Google Code Jam 2011 – Round 3 – user: linguo

Language Popularity

	Problem A		Problem B		Problem C		Problem D		Totals	
Language	S	L	S	L	S	L	S	L	Sets	People ▲
C++	273	256	264	214	162	68	244		1481	317 / 19
Java	48	48	43	31	32	16	45		263	54 / 6
Python	16	15	9	8	4	2	6	1	61	17 / 1
C#	9	9	8	6	5	3	9		49	13

Google Code Jam 2013 – Round 2 – user: bmerry

Language Popularity

	Problem A		Problem B		Problem C		Problem D		Totals	
Language	S	L	S	L	S	L	S	L	Sets	People ▲
C++	1130	617	841	673	298	153			3712	1315 / 393
Java	202	87	134	116	38	16			593	233 / 61
Python	142	70	113	99	10	6	1	1	442	197 / 45
C#	29	8	18	17	6	1			79	37 / 3

Programmazione competitiva 101

“L'obiettivo è scrivere il codice sorgente di programmi che sono in grado di risolvere i problemi presentati. La maggior parte dei problemi sono di natura matematica o logica.”

- Competizioni brevi – alcune ore:
  - International Olympiad in Informatics, ACM International Collegiate Programming Contest, Google Code Jam, Facebook Haker Cup, TopCoder Algorithm Open
  - TopCoder SRM, Sphere Online Judge, Codeforces
- Competizioni lunghe – da alcuni giorni a diversi mesi:
  - TopCoder Marathon Match, Kaggle, Google AI Challenge, Al Zimmermann's Programming Contests

Programmazione competitiva 101

Come funziona una competizione?

- Google Code Jam – 2 ½ ore, posizione data da punteggio e tempo
  - Testo del problema inclusi i limiti e con qualche dataset di input e l'output corrispondente
  - Modellare, programmare, testare, debuggare, ottimizzare...
  - Download del dataset di input e avvio del cronometro
  - Far girare il programma per ottenere l'output
  - Upload dell'output entro 4 minuti
  - L'online judge dichiara l'output corretto o errato
  - Si ottengono punti o si può riprovare con un dataset diverso

Programmazione competitiva 101

Vincoli e risorse

- Vincoli → Tempo di esecuzione e memoria utilizzata
  - CPU (prestazioni e core), RAM (dimensione), Storage (dimensione e prestazioni)
- Risorse → Tempo
  - Modellare
  - Programmare
  - Testare
  - Debuggare
  - Ottimizzare

Problema: trovare tutti i numeri primi fino a N

Definizione: un numero primo è un intero positivo che ha esattamente due divisori distinti: 1 e se stesso

- “Trial division” con tutti i numeri – one-liner buono fino a N=100000:
  - `P = [p for p in range(2,N+1) if all(p%n!=0 for n in range(2,p))]`
- “Trial division” con i primi
- Crivello di Eratostene

> “Sift the Two's and Sift the Three's,
> The Sieve of Eratosthenes.
> When the multiples sublime,
> The numbers that remain are Prime.”
>
> Anonimo

Problema: trovare tutti i numeri primi fino a N

Implementazione in Python puro senza alcuna ottimizzazione:

```
1  def primes_loop(n):
2      P = range(n + 1)
3      P[1] = 0
4      for p in range(2, int(n ** 0.5) + 1):
5          if P[p]:
6              for m in range(p * p, n + 1, p):
7                  P[m] = 0
8      return [p for p in P if p]
```

Implementazione in Python puro con ottimizzazione:

```
1  def primes_assign_list(n):
2      P = range(n + 1)
3      P[1] = 0
4      for p in range(2, int(n ** 0.5) + 1):
5          if P[p]:
6              P[p * p::p] = [0] * (1 + ((n - p * p) // p))
7      return [p for p in P if p]
```

Problema: trovare tutti i numeri primi fino a N

Implementazione con NumPy che ritorna degli **`int`**:

```
1  def primes_numpy(n):
2      P = np.arange(n + 1, dtype='uint32')
3      P[1] = 0
4      for p in range(2, int(n ** 0.5) + 1):
5          if P[p]:
6              P[p * p::p] = 0
7      return [int(p) for p in P if p]
```

Implementazione con NumPy che ritorna dei **`numpy.uint32`**:

```
1  def primes_all_numpy(n):
2      P = np.arange(n + 1, dtype='uint32')
3      P[1] = 0
4      for p in range(2, int(n ** 0.5) + 1):
5          if P[p]:
6              P[p * p::p] = 0
7      return P[P > 0]
```

Confronto di velocità

PyPy

	max	RAM	CPU
primes_loop	56 M	620 Mb	2.3 s
primes_assign_list	56 M	890 Mb	2.5 s
primes_numpy	56 M	290 Mb	2.4 s
primes_all_numpy	56 M	300 Mb	2.5 s

CPython

	max	RAM	CPU
primes_loop	56 M	2.6 Gb	18 s
primes_assign_list	56 M	2.2 Gb	6.8 s
primes_numpy	56 M	330 Mb	6.8 s
primes_all_numpy	56 M	290 Mb	1.4 s

Confronto di dimensione

PyPy

	max	RAM	CPU
primes_loop	320 M	2.9 Gb	13 s
primes_assign_list	320 M	3.3 Gb	33 s
primes_numpy	560 M	2.6 Gb	25 s
primes_all_numpy	560 M	2.8 Gb	26 s

CPython

	max	RAM	CPU
primes_loop	56 M	2.6 Gb	18 s
primes_assign_list	100 M	3.4 Gb	38 s
primes_numpy	560 M	3.1 Gb	68 s
primes_all_numpy	560 M	2.8 Gb	15 s

Guida ai linguaggi di programmazione

Punti di forza di forza e di debolezza

- C++ con Standard Template Library → compilato
  - Modellare, Programmare, Testare, Ottimizzare → Buono
  - Debuggare → Orribile
  - Velocità, Memoria → Eccellente
- Java → interpretato con un JIT
  - Modellare, Ottimizzare → Buono
  - Programmare, Testare, Debuggare → Beh... è Java :-P
  - Velocità, Memoria → Buono

Guida ai linguaggi di programmazione

Punti di forza di forza e di debolezza

- Python senza ottimizzazione → interpretatao senza JIT
  - Modellare, Programmare, Testare, Debuggare → Excellent
  - Velocità, Memoria → Orribile
- Python con ottimizzazione → interpretatao senza JIT
  - Modellare, Programmare, Testare, Debuggare → Variabile tra Cattivo e Buono
  - Velocità, Memoria → Variabile tra Cattivo e Buono
- PyPy → interpretatao con JIT
  - Modellare, Programmare, Testare, Debuggare → Excellent
  - Ottimizzazione → Good
  - Velocità, Memoria → Buono

GCJ 2012 – Round 1A – Problem B. Kingdom Rush

Problem statement: http://goo.gl/DZRMD and analysis: http://goo.gl/ptgyT

```
from sys import stdin
for t in xrange(1, int(stdin.next().strip())+1):
    N, S = int(stdin.next()), [map(int, stdin.next().split()) for i in xrange(N)]
    r, st = 0, 0
    while len(S) > 0:
        T = [s for s in S if s[1]<=st]
        if len(T) > 0:
            S = [s for s in S if s[1]>st]
            r += len(T)
            st += 2 * len(T) - sum(t[0] > 2001 for t in T)
            continue
        A = sorted([s for s in S if s[0]<=st], lambda x,y: y[1]-x[1])
        if len(A) == 0:
            r = 'Too Bad'
            break
        r += 1
        st += 1
        A[0][0] = 2002
    print 'Case #%d: %s' % (t, r)
```

5x faster with PyPy

Al Zimmerman's Programming Contests Factorials

Definizione del problema: http://www.azspcs.net/Contest/Factorials

```
[...]
while len(slp):
        current = slp[-1]
        for i, other in enumerate(slp):
            for j in [0, 1, 2]:
                if j==0:
                    if i==0: continue
                    next = current * other
                elif j==1:
                    if i==len(slp)-1: continue
                    next = current + other
                elif j==2:
                    if i==len(slp)-1: continue
                    next = max(current, other) - min(current, other)
                if lcm % next != 0:
                    continue
                if next in slp or next in nexts[-1]:
                    continue
                if next < current and test_slp_next(slp[:-1], next):
                    continue
[...]
```

5x faster with PyPy

GCJ 2012 – Round 1B
Problema B. Tide Goes In, Tide Goes Out

Definizione del problema: http://goo.gl/7qRzA e analisi: http://goo.gl/JEcSM

```
from sys import stdin
import heapq as hp
for tc in range(1, int(stdin.next())+1):
    H, N, M = map(int, stdin.next().split())
    CH = [map(int, stdin.next().split()) for i in range(N)]
    CL = [map(int, stdin.next().split()) for i in range(N)]
    T = [[2**31]*M for i in xrange(N)]
    T[0][0] = 0.
    F = [(T[0][0], 0, 0)]
    while len(F):
        t, j, i = hp.heappop(F)
        if j==N-1 and i==M-1:
            break
        for jj, ii in [(j-1,i), (j,i-1), (j+1,i), (j,i+1)]:
            if not (0<=jj<N and 0<=ii<M):
                continue
            if min(CH[j][i],CH[jj][ii]) - max(CL[j][i],CL[jj][ii]) < 50:
                continue
            ts = max(t, (H + 50 - CH[jj][ii])/10.)
            if ts > 0.:
                ts += 1. if (H-10*ts-CL[j][i]) >= 20 else 10.
            if ts < T[jj][ii]:
                T[jj][ii] = ts
                hp.heappush(F, (ts, jj, ii))
    print 'Case #%s: %s' % (tc, T[-1][-1])
```

2x SLOWER with PyPy

Setup di PyPy per le competizioni

Preparare un virtualenv con l'ultima release di PyPy con:

- IPython – http://ipython.org/
- PyFlake – https://pypi.python.org/pypi/pyflakes
- NumPyPy – nella distribuzione PyPy
- NetworkX – Grafi – http://networkx.github.io/
- PIL – Image processing – http://www.pythonware.com/products/pil/

Una propria libreria di algoritmi:

- PriorityDictionary – partially ordered dict – http://goo.gl/aWg6r
- Dijkstra Shortest Path Algorithm – http://goo.gl/pQaLo
- GCD, LCM, binom, isqrt, etc...

Limitazioni di PyPy nelle competizioni

Alcune librerie non sono ancora portate su CFFI:

- NumPy – http://www.numpy.org/ (NumPyPy è una riimplementazione parziale)
- SciPy – Scientific computing library – http://www.scipy.org/
- Gmpy2 – Numerical library – http://code.google.com/p/gmpy/

Processi piccoli non performano bene:

- Processi di breve durata soffrono per la lentezza del warm-up
- Processi che utilizzano poca memoria soffrono per il maggiore memory footprint di PyPy

Take away lessons

PyPy: Vantaggi rispetto a Python

- Tempo di Modellazione
  - Possibilità di programmare a basso livello se serve, un po' come il C++
- Tempo per Programmazione e Test
  - Il codice semplice di solito è eseguito con prestazioni ragionevoli
- Tempo di ottimizzazione della performance
  - si può fare a meno di numerose tecniche di ottimizzazione
  - di solito offre buona performance della memoria e buona velocità per task impegnativi
- Tempo di debugging
  - Codice semplice + meno ottimizzazione == debugging più semplice

Grazie.

Alessandro Amici <a.amici@bopen.it>

http://linkedin.com/in/alexamici

search me on Google+

B-Open Solutions – http://bopen.it