Primeritat i factorització

Artur Travesa

(versió 2024-07)

Capítol 0. Un garbell d'Eratòstenes

El garbell d'Eratòstenes és la primera eina que podem utilitzar per a fer llistes de nombres primers. Sovint se'n parla a l'ensenyament bàsic o secundari, tot i que, també sovint, sense justificació.

El procediment bàsic és molt senzill: es tracta de fer una llista de tots els nombres menors que una certa fita i teure'n els compostos d'una manera prou eficient. D'aquesta manera, els nombres que queden a la llista són (els) primers.

A més a més de ser útil per a obtenir llistes de nombres primers, el garbell servirà per a altres algoritmes posteriors de factorització de nombres naturals.

0.0. Una versió bàsica

A partir d'una llista de bits (inicialment, uns) que representen el nombre 2 i els nombres senars des de 3 fins a una certa fita, es tracta de treure (posar a zero) els bits que corresponen a múltiples (no primers) dels nombres primers. Aquests, són els que es corresponen amb els bits que queden sense suprimir.

La teoria en què es fonamenta l'algoritme es pot veure, per exemple, en **[Tr-Ar**, cap. II, §3.6**]**.

La funció següent és una implementació bàsica del garbell d'Eratòstenes,

In [1]:
```
def Eratostenes(ff):
    f=floor((ff+1)/2)
    pr=[1 for i in range(f)]
    i=1
    k=floor((sqrt(ff)+1)/2)
    while i<k:
        if pr[i]==1:
            for j in range(2*i*(i+1),f,2*i+1):
                pr[j]=0
        i=i+1
    return pr

```

Notem que aquesta funció només calcula una llista de zeros i uns, amb un 1 en el primer lloc (el lloc n=0, pensat per al primer 2), un 1 en els llocs n-èsims (n>=1) que corresponen als nombres primers senars 2n+1, i un zero en els altres, que corresponen als nombres compostos senars, 2n+1.

Aquesta llista de zeros i uns hauria de ser suficient per a moltes aplicacions, ja que el valor concret del nombre primer es calcula fàcilment a partir de l'índex del lloc que ocupa el bit corresponent i, en canvi, és més senzill (i, molt important, consumeix menys recursos) emmagatzemar aquesta llista.

0.0.0 Exemples

Per a $f \geq 1$ la funció **Eratostenes(ff)** proporciona una llista no buida.

In [2]:
```
1 Eratostenes(1)
```

Out[2]:
```
[1]
```

In [3]:
```
1 Eratostenes(25)
```

Out[3]:
```
[1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0]
```

In [4]:
```
1 er=Eratostenes(100)
```

In [5]:
```
1 print(er)
```

```
[1, 1, 1, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1,
0, 1, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 1
, 0, 0, 0, 1, 0]
```

In [6]:
```
1 %time er=Eratostenes(10000)
```

```
CPU times: user 2.3 ms, sys: 188 µs, total: 2.49 ms
Wall time: 2.52 ms
```

In [7]:
```
1 %time er=Eratostenes(1000000)
```

```
CPU times: user 301 ms, sys: 8.1 ms, total: 310 ms
Wall time: 309 ms
```

In [8]:
```
1 %time er=Eratostenes(10000000)
```

```
CPU times: user 3.27 s, sys: 152 ms, total: 3.42 s
Wall time: 3.43 s
```

0.1. La llista dels nombres primers

In [9]:
```
def LlistaDePrimers(ff):
    f=floor((ff+1)/2)
    pr=[1 for i in range(f)]
    i=1
    k=floor((sqrt(ff)+1)/2)
    while i<k:
        if pr[i]==1:
            for j in range(2*i*(i+1),f,2*i+1):
                pr[j]=0
        i=i+1
    lta=[pr[n]*(2*n+1) for n in range(f) if pr[n]>0]
    lta[0]=2
    return lta

```

In [10]:
```
LlistaDePrimers(25)
```

Out[10]:
```
[2, 3, 5, 7, 11, 13, 17, 19, 23]
```

In [11]:
```
print(LlistaDePrimers(103))
```

```
[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]
```

In [12]:
```
%time lta=LlistaDePrimers(10000)
```

```
CPU times: user 10.3 ms, sys: 0 ns, total: 10.3 ms
Wall time: 9.87 ms
```

In [13]:
```
len(lta)
```

Out[13]:
```
1229
```

In [14]:
```
%time lta=LlistaDePrimers(1000000)
```

```
CPU times: user 492 ms, sys: 28 ms, total: 520 ms
Wall time: 519 ms
```

In [15]:
```
len(lta)
```

Out[15]:
```
78498
```

In [16]:
```
%time lta=LlistaDePrimers(10000000)
```

```
CPU times: user 5.19 s, sys: 116 ms, total: 5.3 s
Wall time: 5.3 s
```

In [17]:
```
len(lta)
```

Out[17]:
```
664579
```

0.2. Observacions

Sobre la fita

La funció **LlistaDePrimers(ff)** calcula la llista dels nombres naturals primers menors que una fita "raonable" en un temps "raonable". Però notem que la necessitat d'espai és proporcional al valor de la fita; per tant, exponencial en el nombre de bits de la fita. Això fa que les fites "raonables" no puguin ser extraordinàriament grans.

Sobre la programació

Aquesta implementació de les funcions **Eratostenes(ff)** i **LlistaDePrimers(ff)** no és, ni de bon tros, òptima.

D'una banda, una bona implementació hauria de fer servir només un bit per a indicar la posició de cada nombre senar menor que la fita; en canvi, aquesta implementació utilitza, probablement, un mínim d'un byte per a cadascuna d'aquestes posicions (si no n'utilitza més).

A més a més, probablement caldria treballar directament sobre bits i no sobre bytes. Segurament, seria un bon exercici de programació, lluny de l'objectiu que perseguim, fer-ne la programació en llenguatge assemblador; això podria servir, per exemple, per a estudiar la velocitat real dels processadors i el seu comportament en tasques no trivials. Però no insistiré més en aquest fet.

D'altra banda, es fa sempre l'assignació pr[j]=0. Caldria veure si això és més o menys eficient que fer l'assignació condicionada només al cas en què sigui pr[j]=1; és a dir, si és més o menys eficient que utilitzar la comanda if pr[j]==1: pr[j]=0.

D'altra banda, a més a més del càlcul de les posicions que ens permeten dir quins són els nombres primers, la funció **LlistaDePrimers(ff)** calcula explícitament aquests nombres, fet que és, gairebé sempre, innecessari.

Sobre l'emmagatzemament

Notem també que per a emmagatzemar el resultat és molt més costós fer-ho amb la llista dels valors que només amb la successió de bits.

Per exemple, per a emmagatzemar els primers fins a 100, si es contempla una successió de bits, només en calen 50. Però si es volen emmagatzemar els valors en la seva forma més compacta (expressions binàries), i es té en compte que el nombre primer més gran menor que 100 (que és 97) necessita 7 bits, hauríem de fer servir 7 bits per a cadascun dels valors menors (caldria saber on comença i on acaba cada nombre primer) de manera que per als 25 nombres primers necessitaríem 175 bits (tres vegades i mitja més que la llista de bits).

I si es volguessin emmagatzemar els 664579 nombres primers menors que 10000000, i en tenir en compte que el més gran, 9999991, és de 24 bits, necessitaríem 15949896 bits, en comptes dels 5000000 necessaris per a emmagatzemar la llista de bits (ara, gairebé 3.2 vegades més).

Sobre el SageMath

SageMath té implementada la funció prime_range( ) que permet resoldre aquesta tasca

molt ràpidament, i amb més prestacions.

In [18]:
```
1 %time lta1=prime_range(10000000)
```

```
CPU times: user 196 ms, sys: 23.8 ms, total: 220 ms
Wall time: 220 ms
```

In [19]:
```
1 len(lta1)
```

Out[19]:
```
664579
```

Fi del Capítol 0