# Số nguyên tố

Định nghĩa số nguyên tố

Chúng ta nhắc lại một số kiến thức phổ thông về *số nguyên tố*.

* Định nghĩa 1. Số nguyên *p* ≥ 2 là *số nguyên tố* nếu *p* chỉ chia hết cho 1 và chính nó.
* Định nghĩa 2. Số nguyên dương *p* là *số nguyên tố* nếu *p* có đúng hai ước (là 1 và *p*).
* Mệnh đề 1. Tập các số nguyên dương được chia thành ba tập con:
* Tập chứa duy nhất số 1 là số có đúng một ước: {1}.
* Tập các số nguyên tố, mỗi số có đúng hai ước: {2, 3, 5, …}.
* Tập các số còn lại được gọi là *hợp số,* mỗi hợp số có trên hai ước: {4, 6, 8, 9, …}.
* Mệnh đề 2. Số 2 là số nguyên tố duy nhất *nhỏ nhất* và *chẵn* trong tập các số nguyên tố.
* Mệnh đề 3. Mọi số nguyên tố ngoài 2 đều là *số lẻ*.
* Mệnh đề 4. Có *vô hạn* số nguyên tố.
* Mệnh đề 5. Số nguyên *n* > 2 là số *nguyên tố* khi và chỉ khi *n* *không có ước* trong khoảng .
* Mệnh đề 6. Số tự nhiên lẻ *n* > 2 là số *nguyên tố* khi và chỉ khi *n* *không có ước* *lẻ* trong khoảng
* Mệnh đề 7. Số tự nhiên lẻ *n* > 2 là số *nguyên tố* khi và chỉ khi *n* *không có ước nguyên tố* trong khoảng .

được lấy theo cận trên

Ví dụ

Có 25 số nguyên tố dưới 100 là:

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

## Tổng các số nguyên tố

Cho input file NUM.INP gồm nhiều số tự nhiên ghi cách nhau qua dấu cách. Hãy hiển thị tổng các số nguyên tố trong dãy. Kết thúc file là số 0.

Ví dụ

|  |  |  |
| --- | --- | --- |
| NUM.INP | OUTPUT |  |
| 2714  97  113  133  5  237  0 | 215 | 215 = 97 + 113 + 5 |

Thuật toán

Ta vận dụng mệnh đề 6. Nếu *p* lẻ và không có ước lẻ nào trong khoảng 3 đến thì *p* là nguyên tố. Như vậy thuật toán thực hiện tối đa div 2 phép chia dư.

Chương trình

# Tổng các số nguyên tố

def Go(msg = ' ? '):

if input(msg) == '.': exit(0)

def IsPrime(x):

if x < 2: return False

if x in [2,3,5,7,11,13,17,19]: return True

if x % 2 == 0: return False

for u in range(3, int(x\*\*0.5)+1, 2):

if x % u == 0: return False

return True

def ReadInput():

with open('NUM.INP') as f:

d = list(map(int,f.read().strip().split()))

d.pop() # bỏ phần tử 0 cuối dãy

return d

def Run():

print('Các số \* là nguyên tố:')

sum = 0

for x in ReadInput():

s = str(x)

if IsPrime(x):

s += '\*'

sum += x

print(s, end = ' ')

print('\n sum = ', sum)

# APPLICATION

Run()

print(' T h e E n d')

Output

Các số \* là nguyên tố:

2714 97\* 113\* 133 5\* 237

sum = 215

T h e E n d

Độ phức tạp

Kiểm tra tính nguyên tố của một số tự nhiên *p* cần div 2 phép chia dư.

## Sàng Eratosthenes

*Với mỗi số nguyên dương n hãy liệt kê các số nguyên tố không vượt quá n và ghi vào một file PRIMES.DAT mỗi dòng một số.*

Thuật toán

Vì phép nhân và phép cộng được thực hiện đơn giản hơn phép chia, nên Eratosthenes, nhà toán học vĩ đại người Hy Lạp đã đề xuất ý tưởng tổ chức thuật toán tìm toàn bộ các số nguyên tố trong khoảng từ 1 đến giới hạn *n* cho trước. Người đời sau gọi thuật toán này là Sàng Eratosthenes. Bảng dưới đây minh họa hoạt động của thuật toán sàng với n = 100.

|  |  |  |
| --- | --- | --- |
| **Bài giảng của Eratosthenes** | | A close-up of a person's head  Description automatically generatedEratosthenens  276−194 trước CN |
| *Bước 1* | *Trò hãy viết dãy số từ 1 đến 100 trên bảng đất sét của mình. Trò có đủ 100 số từ 1 đến 100 chưa bị xóa trên bảng.* |
| *Bước 2* | *Xóa số 1, vì 1 không phải là số nguyên tố cũng không phải là hợp số. 1 là số đặc biệt trong dãy số tự nhiên.* |
| *Bước 3* | *Tìm số chưa bị xóa tiếp theo. Gọi số đó là i.* |
| *Bước 4* | *Nếu i > 10 thì dừng thuật toán. Toàn bộ các số không bị xóa trên bảng là các số nguyên tố.*  *Nếu i < 10 thì trò thực hiện Bước 5.* |
| *Bước 5*  *Bước 6* | *Xóa các bội số của i kể từ i2 đến 100.*  *Quay lại bước 3.* |

Table

Description automatically generated

Để mô phỏng thuật toán sàng, ta dùng một dãy byte b đánh dấu số nguyên tố: nếu b[i] = 1 thì i là số nguyên tố; ngược lại, khi b[i] = 0 thì i không phải là số nguyên tố.

Chúng ta cài đặt ba hàm tiện ích.

1. Hàm ByteSieve(n) đánh dấu các số nguyên tố nhỏ thua n trong dãy byte b: b[i] = 1 nếu i là số nguyên tố, ngược lại, b[i] = 0 cho biết i không nguyên tố

b = ByteSieve(12) = [0, 0, 1, 1, 0, 1, 0, 1, 0, 0, 0, 1]

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| *i* | *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* | *9* | *10* | *11* |
| b | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 |

1. Hàm Sieve(n) cho ra danh sách các số nguyên tố nhỏ thua n.

p = Sieve(12) = [2, 3, 5, 7, 11]

1. Hàm SieveFile(n, fn) Ghi vào file fn các số nguyên tố nhỏ thua n.

SieveFile(12,'P12.DAT' ) = [2, 3, 5, 7, 11]

|  |
| --- |
| P12.DAT |
| 2  3  5  7  11 |

Chương trình

# Sieve of Eratosthenes

# Đánh dấu các số nhuyên tố < n

# b[i] = 1: i nguyên tố, b[i] = 0: i không nguuyên tố

def ByteSieve(n):

b = bytearray([1]\*n) # gán n phần tử = 1

b[0] = b[1] = 0 # xóa các số 0 và 1

for i in range(4,n,2): b[i] = 0 # xóa các số chẵn 4:n

for i in range(3, int(n \*\* 0.5), 2):

if b[i]: # i nguyên tố

# xóa các bội j = i\*i:n

for j in range(i\*i, n, i): b[j] = 0

return b

# Các số nguyên tố < n

def Sieve(n):

if n < 2: return []

b = ByteSieve(n)

return [2]+[i for i in range(3,len(b), 2) if b[i]]

# Ghi file fn cacs soos nguyeen toos < n

def SieveFile(n, fn):

p = Sieve(n)

with open(fn,'w',encoding = 'utf-8') as g:

for x in p: g.write(str(x)+'\n')

print('\n Total ',len(p))

SieveFile(100, "P100.DAT")

print(' T h e E n d')

OUTPUT

Total 25

T h e E n d

P100.DAT

2

3

5

7

...

97

Độ phức tạp

Thuật toán Sieve(n) đòi hỏi không quá phép nhân các số nguyên (*log* cơ số 2). Ví dụ, nếu n = 210 = 1024 thì log n = 10, log log n = log 10 = 4 (làm tròn), do đó cần 1024×4 = 4096 phép nhân.

# Luyện tập Số nguyên tố

## Số nguyên tố Mersenne

Số nguyên tố có dạng -1 được gọi là số nguyên tố Mersenne. Ví dụ, các số 3, 31, 127 là những số nguyên tố Mersenne. Tìm tổng của các số nguyên tố Mersenne dưới 2 triệu?

## Đếm số nguyên tố

Viết hàm Count(a,b) cho ra số lượng các số nguyên tố trong khoảng a..b.

## Các số nguyên tố quay vòng (Circular primes)

Problem 35 Project Euler

197 được gọi là số nguyên tố quay vòng vì bản thân số đó là nguyên tố và mỗi lần quay vòng số đó một vị trí đều cho ta số nguyên tố: 197, 971, và 719.

Có tất cả mười ba số nguyên tố quay vòng dưới 100 là: 2, 3, 5, 7, 11, 13, 17, 31, 37, 71, 73, 79, và 97.

Có bao nhiêu số nguyên tố quay vòng dưới một triệu?

## Các số nguyên tố cắt được (Truncatable primes)

Problem 37 Project Euler

Số 3797 có tính chất thú vị: bản thân nó là số nguyên tố và những số còn lại sau mỗi lần xóa đi lần lượt từng chữ số bên phải và bên trái số đó thì phần còn lại đều là những số nguyên tố: 3797, 797, 97, 7 sau đó là 379, 37, 3.

Bạn chỉ cần tìm tổng của mười một số nguyên tố có tính chất như trên.

CHÚ Ý: Các số 2, 3, 5, và 7 không được xem là những số nguyên tố cắt được.

## Số nguyên tố thứ i

Số nguyên tố thứ nhầt là 2, thứ hai là 3,... Hãy hiển thị số nguyên tố thứ i ≤ 20000?

Dữ liệu vào text file PNUM.INP chứa các số thứ tự i ghi cách nhau, kết thúc là số 0.

Dữ liệu ra file PNUM.OUT số nguyên tố thứ i với mỗi i, 0 < i ≤ 20000 đọc từ input file.

Ví dụ

|  |  |
| --- | --- |
| PNUM.INP | PNUM.OUT |
| 10  2  397  1001  10001  2000  2017  105  20000  0 | 29  3  2719  7927  104743  17389  17539  571  224737 |

## Số nguyên tố cùng độ cao

Độ cao của một số tự nhiên là tổng các chữ số của số đó. Với mỗi cặp số tự nhiên n và h cho trước hãy liệt kê các số nguyên tố không vượt quá n và có độ cao h, 10 ≤ n ≤ 1000000; 1 ≤ h ≤ 54.

Ví dụ

Prime(1000,16): n = 1000, h = 16. Kết quả: 15 số nguyên tố độ cao 16:

79, 97, 277, 349, 367, 439, 457, 547, 619, 673, 691, 709, 727, 853, 907.

## Số nguyên tố cùng số bít 1

Với mỗi n và h cho trước hãy cho biết có bao nhiêu số nguyên tố không vượt quá n và trong dạng nhị phân chứa đúng h bit 1? 10 ≤ n ≤ 1000000; 1 ≤ h ≤ 30.

Ví dụ

Prime(100, 4)

Có 7 số nguyên tố trong khoảng 1..100

chứa đúng h = 4 bit 1 (trong hệ đếm 2). Đó là các số

23 = 101112, 29 = 111012 , 43 = 1010112 , 53 = 1101012,

71 = 10001112, 83 = 10100112, 89 = 10110012.

## Phân tích ra thừa số nguyên tố

Định lý cơ bản của số học

*Mọi số nguyên dương n* ≥ *2 đều có thể được phân tích thành tích*

*trong đó, pi là các số nguyên tố được sắp tăng chặt, mi là bậc của các pi trong n, 1 ≤ i ≤ k.*

Ví dụ

*Viết hàm p, m = Decompose(n) phân tích số tự nhiên n* ≥ *2 thành hai dãy*

* *dãy các số nguyên tố sắp tăng chặt p và*
* *dãy các số mũ tương ứng m*

Ví dụ

p, m = Decompose(392) cho ta

p = [2,7]

m = [3,2]

p, m = Decompose(392\*97) cho ta

p = [2,7,97]

m = [3,2,1]

## Hàm Phi Euler

*Hàm Phi Euler của số nguyên dương n, ký hiệu là , lá số lượng các số nguyên dương nhỏ thua n và nguyên tố cùng cùng nhau với n.*

trong đó #S là lực lượng (số phần tử) của tập S; (a,b) là ước chung lớn nhát của a và b.

Ví dụ

Hãy viết hàm nay?

## Số Carmichael

Ta biết, theo định lý Fermat nhỏ: nếu n là số nguyên tố thì (\*)

với mọi số nguyên dương x nguyên tố cùng nhau với n.

Có những hợp số cũng có tính trên, cụ thể là số Carmichael là hợp số n thỏa tính chất (\*) với mọi số nguyên dương x nhỏ thua n và nguyên tố cùng nhau với n.

Để ý rằng nếu tính chất (\*) thỏa với mọi số x nhỏ thua n và nguyên tố cùng nhau với n thì tính chất đó cũng thỏa với mọi số x nguyên tố cùng nhau với n. Thật vậy, theo tính chất của hàm ước chung lớn nhất, ta có (x, n) = (x mod n, n) = 1 và x mod n < n, do đó .

Hãy liệt kê các số Carmichael dưới 40000.

## Maximum Prime

Cho một số nguyên dương x dài tối đa 200 chữ số. Nhiệm vụ của bạn là tìm một đoạn gồm các chữ số liên tiếp nhau trong x sao cho đoạn này tạo thành một số nguyên tố lớn nhất. Dữ liệu vào gồm nhiều test, mỗi test chiếm một dòng: Mỗi dòng chứa một số dài tối đa 200 chữ số. Kết thúc bởi dòng chứa số 0 (không cần xử lý).

Kết qủa: Với mỗi test, in ra một số nguyên tố lớn nhất dài không quá 6 chữ số.

Ví dụ

|  |  |  |
| --- | --- | --- |
| MAXPRIME.INP | OUTPUT | Giải thích |
| 11245 | 11 | 11245 |
| 91321150448 | 1321 | 91321150448 |
| 1226406 | 2 | 1226406 |
| 372441465 | 2441 | 372441465 |
| 2001368741189 | 41189 | 2001368741189 |
| 505099997967743 | 97967 | 505099997967743 |
| 0 |  |  |

# Bài giải Số nguyên tố

## Số nguyên tố Mersenne

Thuật toán

* Nhận xét 1. Nếu 2*n* − 1 là nguyên tố thì *n* nguyên tố. Thật vậy, ta vận dụng hằng đẳng thức

Giả sử *n* là hợp số, tức là *n = uv, u,v* > 1. Khi đó .

**.** Do u, v > 1nên, mâu thuẫn với tính nguyên tố.

* Nhận xét 2. nên ta kiểm tra các số -1 trong sàng Sieve(2M), M = 1000000 (một triệu).

Sau khi gọi thuật toán sàng Sieve(2M) để đánh dấu các số nguyên tố trong khoảng 2..2M vào dãy byte b. Ta biết số Mersenne đầu tiên là . Ta tìm tiếp các số Mersenne với các số nguyên tố n trong khoảng 3..log(2M). Vì các số nguyên tố > 2 đều là các số lẻ nên ta chọn bước tăng là 2.

Để tính logarit nguyên cơ số b của số n, ta vận dụng tính chất

Log(b,n) = 0 nếu n < b

Log(b, x\*y) = Log(b, x) + Log(b, y). Ta có

Log(b, n) = Log(b, b\*(n//b)) = Log(b,b) + Log(b, n//b) = 1 + Log(b, n//b)

Bạn cũng có thể gọi hàm log2 trong thư viện math của Python:

from math import log2

int(log2(x))

Chương trình

# Các số Mersenne < 2M (hai triệu)

from time import time

MN = 2000000

# các số nguyên tố < n

def ByteSieve(n):

b = bytearray([1]\*n)

b[0] = b[1] = 0

# xoa cac so chan ke tu 4

for i in range(4, n, 2): b[i] = 0

for i in range(3, int(n\*\*0.5)+1, 2):

if b[i]: # i prime

# xóa các bội (i\*i, n, i)

for j in range(i\*i, n, i): b[j] = 0

return b

def Log(base, n): # int log

return 1 + Log(base, n // base) if n > base-1 else 0

# Mersenne numbers

def Mersenne():

b = ByteSieve(MN)

sum = 3 # 2^2-1 = 3 số Mersenne đầu tiên

print(3, end = ' ')

for n in range(3, Log(2, MN), 2):

if b[n]: # n prime

p = (1 << n) - 1 # q = 2\*\*n - 1

if b[p]: # p prime

print(p, end = ' ')

sum += p

print('\n sum = ', sum)

# APPLICATION

t = time()

Mersenne()

print(time()-t)

print(' T h e E n d')

Output

3 7 31 127 8191 131071 524287

sum = 663717

1.0833587646484375

T h e E n d

Độ phức tạp

Tương đương với Sieve.

## Đếm số nguyên tố

Thuật toán

1. b = ByteSieve(MN), MN = 2M

2. Hiển thị các số nguyên tố trong sàng a..b

Nếu b la danh sách thì b[i:j] là danh sách con của b

gồm các phần tử từ b[i] đến b[j-1].

Nếu i ≥ j thì b[i:j] = [] (danh sách rỗng).

b.count(1) đếm số giá trị 1 trong danh sách b.

Chương trình

# Đếm các số nguyên tố trong khoảng a..b

from time import time

MN = 2000000

# Các số nguyên tố < n

def ByteSieve(n):

b = bytearray([1]\*n)

b[0] = b[1] = 0

# xóa các số chẵn 4..n

for i in range(4, n, 2): b[i] = 0

for i in range(3, int(n \*\* 0.5) + 1, 2):

if b[i]: # i prime

# xóa các bội (i\*i, n, i)

for j in range(i\*i, n, i): b[j] = 0

return b

# APPLICATION

t = time()

b = ByteSieve(MN)

print(b[2:100].count(1)) # 25

print(b[2:1000000].count(1)) # 78498

print(time()-t)

print(' T h e E n d')

Output

25

78498

1.0624666213989258

T h e E n d

Độ phức tạp

Tương đương với Sieve.

## Các số nguyên tố quay vòng (Circular primes)

Thuật toán

Cho giới hạn MN = 1000000 (1M). Sau khi gọi thuật toán Sàng Eratosthenes Sieve(MN) để đánh dấu các số nguyên tố dưới 1M vào một mảng b[] ta xét từng số nguyên tố *p* với *k* chữ số. Nếu *k −* 1 lần quay đều nhận được số nguyên tố thì số *p* sẽ được chấp nhận. Dĩ nhiên ta chấp nhận 13 số nguyên tố quay vòng dưới 100 đầu tiên, nên chỉ cần tiếp tục kiểm tra từ số nguyên tố lớn hơn 97 trở đi.

Ta cũng lưu ý khi kiểm tra các chữ số trong *p*. Nếu *p* chứa các chữ số chẵn hoặc số 5 thì sau một số lần quay các chữ số này sẽ chuyển về hàng đơn vị và sẽ tạo ra một hợp số.

Ví dụ

1. *p* = 571 dù là số nguyên tố nhưng sau khi quay sẽ sinh ra hợp số 715.

2. *p* = 761 dù là số nguyên tố nhưng sau khi quay sẽ sinh ra hợp số 176.

3. *p* = 101 dù là số nguyên tố nhưng sau khi quay sẽ sinh ra hợp số 110.

Tóm lại, nếu *p* chứa một trong các chữ số trong tập BADDIG = {0, 2, 4, 5, 6, 8} thì ta bỏ qua**.**

Phương án 1

Để quay số x trước hết ta tính số chữ số của x: lenx và tính p10 = 10lenx-1.

Khi thực hiện một lần quay ta chuyển chữ số đơn vị về đầu: (x%10)\*p10 + (x//10).

Ví dụ x = 13973, lenx = 5, p10 = 10lenx-1 = 104 = 10000. Khi đó 13973 → 3\*10000+1397=31397.

Mỗi lần quay ta kiểm tra chữ số đơn vị và số mới sinh.

Phương án 2

Một cải tiến quan trọng là như sau. Giả sử bạn xét đến số x1 = 11939. Số này và 4 số quay của nó đều là số nguyên tố, cụ thể như sau:

x1 = 11939, x2 = 91193, x3 = 39119, x4 = 93911, x5 = 19391

Nhận xét này gợi ý cho bạn đánh dấu 5 số quay của số 11939 trong sàng các số nguyên tố b, tức là gán lại b[xi] = 0, i = 1..5, do đó khi gặp lại các số x2..x5 bạn không phải xét đến chúng nữa. Như vậy bạn tiết kiệm được việc xét 4 số quay tiếp theo là x2..x5, mỗi số bạn tiết kiệm được 4 phép quay, tổng cộng bạn tiết kiệm được 16 phép quay.

Sau khi bạn gọi phương án 2 thì sàng b sẽ bị phá hỏng. Để tránh việc này, bạn cấn dùng thêm một bản sao e của b. Lưu ý rằng câu lệnh e = b trong Python đồng nhất danh sách e với danh sách b, theo nghĩa, nếu e bị thay đổi thì b cũng thay đổi theo. Bạn phải dùng lệnh e = b.copy().

Chương trình

# Circular Primes

# Answer: 55

from time import time

MN = 1000000

P10 = [1,10,100,1000,10000,100000,1000000] # 10^i, i = 0..6

# các chữ số xấu: 0, 2, 4, 5, 6, 8,

BADDIG = [1, 0, 1, 0, 1, 1, 1, 0, 1, 0]

b = bytearray([1]\*MN)

# Các số nguyên tố < n

def ByteSieve(n):

global b

b[0] = b[1] = 0

# xóa các số chẵn 4..n

for i in range(4, n, 2): b[i] = 0

for i in range(3, int(n \*\* 0.5) + 1, 2):

if b[i]: # i prime

# xóa các bội (i\*i, n, i)

for j in range(i\*i, n, i): b[j] = 0

# Phương án 1: x prime

# quay x len-1 lần đều cho số nguyên tố

def AllRot1(x):

n = len(str(x))

head = P10[n-1]

# quay n-1 lần

for i in range(1, n):

c = x % 10 # chu so thu i

if BADDIG[c]: return 0

# quay

x = head\*c + (x // 10)

if not b[x]: return 0

return 1

def Circl1():

c = 13 # count

for x in range(113, MN, 2):

if b[x]: # x prime

if AllRot1(x):

c += 1

print(x, end = ' ')

if c % 10 == 0: print()

print('\n Total:',c)

# Phương án 2: x prime

# xóa các số quay

def AllRot2(e,x):

n = len(str(x))

head = P10[n-1]

# quay n-1 lan

for i in range(1, n):

c = x % 10 # chu so thu i

if BADDIG[c]: return 0

# quay 1 lan

x = head\*c + (x // 10)

if e[x]: e[x] = 0

else: return 0

return n

def Circl2(e):

c = 13 # count

for x in range(113, MN, 2):

if e[x]: c += AllRot2(e,x) # x nguyen to

print(' Total:',c)

# APPLICATION

t = time()

ByteSieve(MN)

print('Time for Sieve:', time()-t)

t = time()

print('Ver 1')

Circl1()

print('Time for Ver 1:', time()-t)

t = time()

print('Ver 2')

e = b.copy()

Circl2(e)

print('Time for Ver 2:', time()-t)

print(' T h e E n d')

Output

Time for Sieve: 0.5624749660491943

Ver 1

113 131 197 199 311 337 373

719 733 919 971 991 1193 1931 3119 3779 7793

7937 9311 9377 11939 19391 19937 37199 39119 71993 91193

93719 93911 99371 193939 199933 319993 331999 391939 393919 919393

933199 939193 939391 993319 999331

Total: 55

Time for Ver 1: 0.5468602180480957

Ver 2

Total: 55

Time for Ver 2: 0.5312325954437256

T h e E n d

Độ phức tạp

Thuật toán gồm hai pha. Pha 1: tương đương với Sieve(n), n log log n. Pha 2: Quay vòng mỗi số x cần len(x) thao tác. Độ phức tạp khi đó sẽ là max(n log log n, mk), k là số lượng các số nguyên tố dưới n; k xấp xỉ đại lượng n/loge(n), m là chiều dài trung bình của các số nguyên dưới n.

## Các số nguyên tố cắt được (Truncatable primes)

Thuật toán

Ta có thể bắt đầu bằng những số dễ nhận biết trong khoảng dưới một trăm. Với các số hai chữ số dạng *ab*, ta đặt nhát cắt | giữa các chữ số để cắt mỗi số thành hai phần *a*|*b*. Nếu cả hai phần *a* và *b* đều là những số nguyên tố thì ta đưa *ab* vào tổng. Ta có

Số 13 cho ta 1|3. Do 1 không phải là số nguyên tố nên 13 bị loại. Nhận xét này cho phép ta loại mọi số nguyên tố có các chữ số thuộc đầu trái hoặc đầu phải không phải là số nguyên tố (một chữ số). Đó là những chữ số nằm trong tập {0, 1, 4, 6, 8, 9}.

Ta xét tiếp.

Số 23 cho ta 2|3 có 2 và 3 đều nguyên tố, ta đưa 23 vào tổng. Bằng lập luận tương tự ta thu được kết quả của pha đầu tiên như sau:

sum = 23 + 37 + 53 + 73 = 186 (tổng của 4 số dưới 100).

Trước hết ta dự đoán rằng trong khoảng dưới một triệu sẽ chứa đủ 11 số nguyên tố cần tìm, do đó ta gọi thuật toán Sàng Eratosthenes Sieve(1000000) để tìm và đánh dấu các số nguyên tố trong khoảng này. Sau đó ta duyệt các số nguyên tố từ 203 cách nhau 2 đơn vị để lấy những số nguyên tố cắt được đưa vào tổng và dừng chương trình khi đã lấy đủ 11 số.

Giả sử *x* là một số nguyên tố có *m* chữ số trong sàng và 1 ≤ *m* ≤ *k*. Khi đó bằng cách dịch chuyển nhát cắt *m*−1 lần, mỗi lần một chữ số ta có thể kiểm tra *x* có được chấp nhận hay không.

Ví dụ

*x* = 739397 là số trong sàng, tức là *x* là số nguyên tố. Vì *x* có *k* = 6 chữ số nên ta đặt nhát cắt 5 lần như sau:

Nhát cắt 1 : 73939|7 cho ta *x* div 10 = 73939 và *x* mod 10 = 7 đều nguyên tố.

Nhát cắt 2 : 7393|97 : cho ta *x* div 100 = 7393 và *x* mod 100 = 97 đều nguyên tố.

Nhát cắt 3 : 739|397 : cho ta *x* div 1000 = 739 và *x* mod 1000 = 397 đều nguyên tố.

Nhát cắt 4 : 73|9397 : cho ta *x* div 10000 = 73 và *x* mod 10000 = 9397 đều nguyên tố.

Nhát cắt 5 : 7|39397 : cho ta *x* div 100000 = 7 và *x* mod 100000 = 39397 đều nguyên tố.

Vậy *x* = 739397 qua được các test và do đó được chấp nhận.

Ví dụ trên cho ta thấy:

Nhát cắt thứ *i* sẽ tách *x* thành hai số  .

Số bên trái là

Số bên phải là

1 ≤ *i* < *k*

Hàm Cut(*x*) sẽ cho ra giá trị True nếu mọi nhát cắt của số *x* đều tạo ra hai số nguyên tố; ngược lại, hàm cho ra giá trị False.

Các chương trình dưới đây cho kết quả là tổng của 11 số nguyên tố cắt được:

sum = 23 + 37 + 53 + 73 + 313 + 317 + 373 + 797 + 3137 + 3797 + 739397 = 748317.

Ta cũng tính sẵn các giá trị để tiện dùng.

Chương trình

"""

Truncatable primes

Answer: 748317

sum = 23 + 37 + 53 + 73 + 313 + 317 + 373

+ 797 + 3137 + 3797 + 739397

= 748317.

"""

from time import time

MN = 1000000

POW10 = [10, 100, 1000, 10000, 100000, 1000000] # 10^i, i = 1..6

# Các số nguyên tố < n

def ByteSieve(n):

b = bytearray([1]\*n)

b[0] = b[1] = 0

# xóa các số chẵn 4..n

for i in range(4, n, 2): b[i] = 0

for i in range(3, int(n \*\* 0.5) + 1, 2):

if b[i]: # i prime

# xóa các bội (i\*i, n, i)

for j in range(i\*i, n, i): b[j] = 0

return b

def Cut(b, x):

for p in POW10:

thuong, du = divmod(x, p)

if thuong == 0: break

if not b[thuong]: return False

if not b[du]: return False

return True

def Truncat(b):

sum = 23 + 37 + 53 + 73

print(23, 37, 53, 73, end = ' ')

c = 4 # counter

for x in range(203, MN , 2):

if b[x]:

if Cut(b, x):

print(x, end = ' ')

sum += x

c += 1

if c == 11: break

print('\n Sum = ', sum)

# APPLICATION

t = time()

b = ByteSieve(MN)

Truncat(b)

print(time()-t)

print(' T h e E n d')

Output

23 37 53 73 313 317 373 797 3137 3797 739397

Sum = 748317

0.8280997276306152

T h e E n d

Độ phức tạp

Tương đương với bài các số nguyên tố quay vòng.

## Số nguyên tố thứ i

Thuật toán

**Nếu liệt kê mười số nguyên tố đầu tiên là 2, 3, 5, 7, 11, 13, 17, 19, 23 và 29 ta thấy số nguyên tố thứ sáu là 13. Ta cần tìm số nguyên tố thứ i.**

Số tự nhiên lẻ *x* > 2 là *nguyên tố* nếu và chỉ nếu *x* không có ước nguyên tố nào ≤ int. Ta sẽ tạo ra dãy *k* số nguyên tố đầu tiên và ghi vào một danh sách p[] sau đó thêm dần các số nguyên tố vào p như sau.

Bước 1. *Khởi trị*: Đặt *p*[1..10] là mười số nguyên tố đầu tiên.

**Bước 2. Lặp: Giả sử p = (p[1], p[2], ..., p[k]), k ≥ 2 là** *k* số nguyên tố trong dãy p và **p[n] là số đầu tiên trong dãy p thỏa điều kiện . Để ý rằng mọi số nguyên tố lớn hơn 2 đều là số lẻ. Với mọi số lẻ x trong khoảng , nếu x không chia hết cho mọi số nguyen tố p[1..n] thì x là nguyên tố. Ta thêm x vào cuối dãy p. Nếu k = i thì ta return p[k] và dừng thuật toán, ngược lại, ta tiếp tục lặp.**

**Ví dụ dưới đây minh họa tiếp cận này.**

**Ví dụ**

**Cho k = 10, n = 4 và p[1..10] = (2, 3, 5, 7, 11, 13, 17, 19, 23, 29) là mười số nguyên tố đầu tiên đã được sinh ra. Ta có, p[n] = p[4] = 7, . Với mỗi số lẻ x từ 29+2 = 31 đến 49−2 = 47, ta kiểm tra xem các số lẻ 31, 33, 35, 37, 39, 41, 43, 45 và 47 để tìm ra những số không chia hết cho các số p[2..3] = (3, 5) rồi nạp vào cuối dãy p:**

* **x = 31. Vì 3 và 5 không là ước của 31, 31 là số nguyên tố mới. Ta thêm 31 vào cuối dãy p: p[11] = 31**
* **x = 33. Vì 3 là ước của 33, bỏ qua**
* **x = 35. Vì 5 là ước của 35, bỏ qua**
* **x = 37. Vì 3 và 5 không là ước của 37, 37 là số nguyên tố mới. Ta thêm 37 vào cuối dãy p: p[12] = 37**
* **x = 39. Vì 3 là ước của 39, bỏ qua**
* **x = 41. Vì 3 và 5 không là ước của 41, 41 là số nguyên tố mới. Ta thêm 41 vào cuối dãy p: p[13] = 41**
* **x = 43. Vì 3 và 5 không là ước của 43, 43 là số nguyên tố mới. Ta thêm 43 vào cuối dãy p: p[14] = 43**
* **x = 45. Vì 5 là ước của 45, bỏ qua**
* **x = 47. Vì 3 và 5 không là ước của 47, 47 là số nguyên tố mới. Ta thêm 47 vào cuối dãy p: p[15] = 47.**

**Đến đây ta thu được k = 15, và p[1…15] = (2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 41, 43, 47), trong đó các số nguyên tố mới thêm được in đậm. Tiến trình sẽ được lặp với n = 5, k = 15 và kiểm định các số lẻ trong khoảng 47+2..(p[n])2 – 2 = 121 − 2 = 119.**

**Input của bài gồm nhiều số nên ta dùng kỹ thuật *tái sử dụng* như sau. Với mỗi số i đọc từ input file ta xét một trong hai trường hợp :**

* **Nếu i ≤ k ta return p[i]**
* **Nếu i > k  ta tìm tiếp các số nguyên tố từ thứ k+1 đến thứ i và return p[i].**

Chương trình

# Số nguyên tố thứ i

from time import time

FN = "PNUM.INP" # input

GN = "PNUM.OUT" # output

n = 3

# 10 số nguyên tố đầu tiên

p = [0, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29]

# x nguyên tố khi và chỉ khi x lẻ và

# không có ước nguyên tố p[2..n]

def IsPrime(x):

for i in range(2, n+1):

if x % p[i] == 0: return False

return True

def ReadInput():

with open(FN) as f:

inp = list(map(int, f.read().split()))

inp.pop() # bỏ phần tử 0 cuối dãy

return inp

# Tìm đến số nguyên tố thứ i

def Find(i):

global n, p

while len(p) <= i:

n += 1

pp = p[n]\*p[n]

# duyệt từ cuối p đến pp-1

for x in range(p[-1]+2, pp, 2):

if IsPrime(x): p.append(x)

return p[i]

def Run():

inp = ReadInput()

print(' Init p: ', p)

print(' Trong kho hiện có', len(p) - 1, ' số NT')

print(' Cần tìm các số NT thứ: \n', inp)

for i in inp:

print(' Số NT thứ ',i, '?', end = ' ')

if i < len(p):

print(' có sẵn trong kho:', p[i])

else:

x = Find(i)

print(' Tìm được', x)

print(' Trong kho hiện có', len(p) - 1, ' số NT')

# write result

with open(GN, 'w') as g:

for i in inp:

g.write(str(p[i])+'\n')

# APPLICATION

t = time()

Run()

print(time()-t)

print(' T h e E n d')

Output

Init p: [0, 2, 3, 5, 7, 11, 13, 17, 19, 23, 29]

Trong kho hiện có 10 số NT

Cần tìm các số NT thứ:

[10, 2, 397, 1001, 10001, 2000, 2017, 105, 20000]

Số NT thứ 10 ? có sẵn trong kho: 29

Số NT thứ 2 ? có sẵn trong kho: 3

Số NT thứ 397 ? Tìm được 2719

Trong kho hiện có 409 số NT

Số NT thứ 1001 ? Tìm được 7927

Trong kho hiện có 1163 số NT

Số NT thứ 10001 ? Tìm được 104743

Trong kho hiện có 10415 số NT

Số NT thứ 2000 ? có sẵn trong kho: 17389

Số NT thứ 2017 ? có sẵn trong kho: 17539

Số NT thứ 105 ? có sẵn trong kho: 571

Số NT thứ 20000 ? Tìm được 224737

Trong kho hiện có 20385 số NT

1.1718404293060303

T h e E n d

Chú thích

p.pop: bỏ phần tử cuối cùng trong danh sach p

p[-1]: giá trị của phần tử cuối cùng trong danh sach p

Khi gọi hàm Find để tìm số nguyên tố thứ 2000 ta đã chạy quá chỉ số 2000, do đó khi cần tìm số nguyên tố thứ 2017 ta lấy sẵn trong kho.

Độ phức tạp

Tương đương với Sieve

## Số nguyên tố cùng độ cao

Thuật toán

Sau khi gọi Sieve(maxn), ta duyệt tìm các số nguyên tố p có độ cao H(p) = h.

Chương trình

# Các số nguyên tố cùng độ cao

from time import time

# primes < n

def ByteSieve(n):

b = bytearray([1]\*n)

b[0] = b[1] = 0

# xoa cac so chan ke tu 4

for i in range(4,n,2): b[i] = 0

for i in range(3, int(n \*\* 0.5) + 1, 2):

if b[i]: # neu i nguyen to

# xoa cac boi cua i ke tu i^2

for j in range(i\*i, n, i): b[j] = 0

return b

def H(x): return sum([int(c) for c in str(x)])

# các số nguyên tố < n, độ cao h

def Prime(b, n, h):

return [i for i in range(2,n) if b[i] and H(i) == h]

# APPLICATION

t = time()

mn = 1000000

b = ByteSieve(mn) # các số nguyên tố < 1M

n = 1000

for h in range(10, 17):

print('Các số nguyên tố <',n, 'độ cao',h,':')

d = Prime(b, n, h)

m = len(d)

if m == 0: print(None)

else:

m2 = m // 2

print(d[:m2])

print(d[m2+1:])

print('Total', m)

print(time()-t)

print(' T h e E n d')

Output

Các số nguyên tố < 1000 độ cao 10 :

[19, 37, 73, 109, 127, 163, 181]

[307, 433, 523, 541, 613, 631, 811]

Total 15

Các số nguyên tố < 1000 độ cao 11 :

[29, 47, 83, 137, 173, 191, 227, 263]

[317, 353, 443, 461, 641, 821, 911]

Total 16

Các số nguyên tố < 1000 độ cao 12 :

None

Các số nguyên tố < 1000 độ cao 13 :

[67, 139, 157, 193, 229, 283, 337, 373]

[463, 571, 607, 643, 661, 733, 751, 823]

Total 17

Các số nguyên tố < 1000 độ cao 14 :

[59, 149, 167, 239, 257, 293, 347, 383]

[491, 509, 563, 617, 653, 743, 761, 941]

Total 17

Các số nguyên tố < 1000 độ cao 15 :

None

Các số nguyên tố < 1000 độ cao 16 :

[79, 97, 277, 349, 367, 439, 457, 547]

[673, 691, 709, 727, 853, 907]

Total 15

0.54561737060546875

T h e E n d

Độ phức tạp

Duyệt k số nguyên tố p, mỗi số p tính độ cao cần len(p) phép chia.

## Số nguyên tố cùng số bít 1

Bài này tương tự bài trước vì trong dạng nhị phân (chỉ chứa các chữ số 0/1) thì tổng các chữ số chính là số bit 1. Hàm H(x, base=10) tính tổng các chữ số của số tự nhiên x khi biểu diễn x theo hệ đếm base > 1 bất kỳ. Như vậy, để tính tổng các chữ số của x trong hệ đếm 2 ta gọi H(x,base = 2).

Chương trình

# Các số nguyên tố cùng số bit 1

from time import time

DIGIT = '0123456789'

# primes < n

def ByteSieve(n):

b = bytearray([1]\*n)

b[0] = b[1] = 0

# xóa các số chẵn ≥ 4

for i in range(4,n,2): b[i] = 0

for i in range(3, int(n \*\* 0.5) + 1, 2):

if b[i]: # i nguyên tố

# xóa các bội j của i ≥ i^2

for j in range(i\*i, n, i): b[j] = 0

return b

def H(x, base = 10): # độ cao của x trong base

h = 0 # digit sum

while x > 0:

x, r = divmod(x, base)

h += r

return h

def Str(x, base = 10): # int -> string

if x == 0: return '0'

s = ''

while x > 0:

x, r = divmod(x, base)

s = DIGIT[r] + s

return s

# các số nguyên tố < n, độ cao h

def Prime(b, n, h, base = 10):

return [i for i in range(2,n) if b[i] and H(i, base) == h]

# APPLICATION

t = time()

mn = 1000000

b = ByteSieve(mn) # các số nguyên tố < 1M

n, base = 100, 2

for h in range(2,6):

print('Các số nguyên tố <',n, 'hệ đếm', base,'độ cao', h,':')

d = Prime(b, n, h, base)

print(d)

print([Str(x,2) for x in d])

print('Total', len(d))

print(time()-t)

print(' T h e E n d')

Output

Các số nguyên tố < 100 hệ đếm 2 độ cao 2 :

[3, 5, 17]

['11', '101', '10001']

Total 3

Các số nguyên tố < 100 hệ đếm 2 độ cao 3 :

[7, 11, 13, 19, 37, 41, 67, 73, 97]

['111', '1011', '1101', '10011', '100101', '101001', '1000011', '1001001', '1100001']

Total 9

Các số nguyên tố < 100 hệ đếm 2 độ cao 4 :

[23, 29, 43, 53, 71, 83, 89]

['10111', '11101', '101011', '110101', '1000111', '1010011', '1011001']

Total 7

Các số nguyên tố < 100 hệ đếm 2 độ cao 5 :

[31, 47, 59, 61, 79]

['11111', '101111', '111011', '111101', '1001111']

Total 5

0.4999854564666748

T h e E n d

Độ phức tạp

Duyệt k số nguyên tố p, mỗi số p tính độ cao cần m = 32 phép chia đôi cho các số nguyên.

Phương án 2

Có thể dùng hàm bin để giải bài này như sau.

Hàm bin chuyển đổi số sang hệ đếm 2. Ví dụ

bin(13) = '0b1101'

Hai ký tự đầu là '0b’ cho biết đây là số dạng nhị phân. Khi đó

bin(x).count('1')

sẽ cho ta số bit 1 trong số x dạng nhị phân.

Chương trình

# Các số nguyên tố cùng số bit 1

# phương án 2: dùng hàm bin

from time import time

def Go():

if input(' ? ') == '.': exit(0)

# primes < n

# def ByteSieve(n):...

# các số nguyên tố < n, độ cao

def Prime(b, n, h):

return [i for i in range(2,n) if b[i] and bin(i).count('1') == h]

# APPLICATION

t = time()

mn = 1000000

b = ByteSieve(mn) # các số nguyên tố < 1M

n, base = 100, 2

for h in range(2,6):

print('Các số nguyên tố <',n, 'hệ đếm', base,'độ cao', h,':')

d = Prime(b, n, h)

print(d)

print([bin(x) for x in d])

print('Total', len(d))

print(time()-t)

print(' T h e E n d')

Output

Các số nguyên tố < 100 hệ đếm 2 độ cao 2 :

[3, 5, 17]

['0b11', '0b101', '0b10001']

Total 3

Các số nguyên tố < 100 hệ đếm 2 độ cao 3 :

[7, 11, 13, 19, 37, 41, 67, 73, 97]

['0b111', '0b1011', '0b1101', '0b10011', '0b100101', '0b101001', '0b1000011', '0b1001001', '0b1100001']

Total 9

Các số nguyên tố < 100 hệ đếm 2 độ cao 4 :

[23, 29, 43, 53, 71, 83, 89]

['0b10111','0b11101','0b101011','0b110101', '0b1000111', '0b1010011', '0b1011001']

Total 7

Các số nguyên tố < 100 hệ đếm 2 độ cao 5 :

[31, 47, 59, 61, 79]

['0b11111', '0b101111', '0b111011', '0b111101', '0b1001111']

Total 5

0.5468578338623047

T h e E n d

## Phân tích ra thừa số nguyên tố

Thuật toán

Đầu tiên ta viết hàm Degree(n, p) tính bậc của số nguyên tố p trong n, tức là tìm số mũ cao nhất m để là ước của n. Sau lời gọi hàm, giá trị của n sẽ giảm xuống còn

Ví dụ

m, n = Degree(45, 3) cho ta m = 2, n = 5 vì 45 = ×5.

Tiếp đến ta tính các Degree(n, p) với p là các số nguyên tố 2...int. Chú ý rằng sau lần chia cuối cùng ta có thể thu được n = 1 hoặc n là số nguyên tố.

# Bậc của thừa số nguyên tố p trong n

def Degree(n,p): # m,n = Degree(n, p)

m = 0

while n % p == 0:

m += 1

n //= p

return m, n

Chương trình

# Địnn lý cơ bản của Số học

# Phân tích ra thừa số nguyên tố

from time import time

def Go():

if input(' ? ') == '.': exit(0)

# Bậc của thừa số nguyên tố p trong n

def Degree(n,p): # m,n = Degree(n, p)

m = 0

while n % p == 0:

m += 1

n //= p

return m, n

# Decompose: Phân tích ra thừa số nguyên tố

def Decompose(n):

p, m = [], []

if n < 2:

return p, m

# xét riêng các số chẵn

v = 2

d, n = Degree(n, v)

if d > 0:

p.append(v)

m.append(d)

for v in range(3, int(n\*\*0.5)+1, 2):

d, n = Degree(n, v)

if d > 0:

p.append(v)

m.append(d)

if n > 1:

p.append(n)

m.append(1)

return p, m

def Show(n, p, m):

s = str(n) + ' = ' + str(p[0]) + '^' + str(m[0])

for i in range(1, len(p)):

s += ' \* ' + str(p[i])+'^'+str(m[i])

print(s)

def Run():

v = 392

d = [v, v\*97, 2, 103, 97234]

for n in d:

p, m = Decompose(n)

Show(n, p, m)

# APPLICATION

t = time()

Run()

print(time()-t)

print(' T h e E n d')

Output

392 = 2^3 \* 7^2

38024 = 2^3 \* 7^2 \* 97.0^1

2 = 2^1

103 = 103^1

97234 = 2^1 \* 61^1 \* 797.0^1

T h e E n d

Độ phức tạp

.

## Hàm Phi Euler

Thuật toán

Phương án 1: Duyệt tuần tự theo công thức (\*)

Phương án 1 đòi hỏi n lần gọi hàm Gcd.

Các tính chất của hàm Phi Euler

* Nếu n là số nguyên tố thì Ví dụ
* Tổng quát, nếu , thì

Ví dụ

* Nếu a và b là hai số nguyên tố cùng nhau, (*a,b*) = 1 thì .

Ví dụ

Phương án 2

**Công thức:**

Độ phức tạp: cần lần lặp

**Trước hết ta tính các ước nguyên tố khác nhau của n, sau đó tính**  theo (\*\*\*)

Chương trình

# Phi Euler Function

from time import time

def Go():

if input(' ? ') == '.': exit(0)

def Gcd(a, b):

return a if b == 0 else Gcd(b, a % b)

# Phi1: phương án 1

def Phi1(n):

if n < 2: return 0

d = 1

for x in range(2, n):

if Gcd(x,n) == 1: d += 1

return d

# các ước nguyên tố khác nhau của n

def PrimeFactor(n):

p = []

if n < 2: return p # n = 0 | 1

if n % 2 == 0:

p.append(2)

while n % 2 == 0: n //= 2

for i in range(3, int(n\*\*0.5) + 1, 2):

if n % i == 0:

p.append(i)

while n % i == 0: n //= i

if n > 1: p.append(n)

return p

def Phi2(n):

r = n

for p in PrimeFactor(n): r = (r//p)\*(p-1)

return r

def Run():

nn = [7, 29, 392, 200]

for n in nn: print(n, Phi1(n), Phi2(n))

# APPLICATION

t = time()

Run()

print(time()-t)

print(' T h e E n d')

Output

7 6 6

29 28 28

392 168 168

200 80 80

0.015624046325683594

T h e E n d

## Số Carmichael

Thuật toán

Để ý rằng nếu x là số chẵn thì ucln(x, 2) = 2, do đó x không thể là số Carmichael.

Phương án 1

Duyệt các hợp số lẻ và kiểm tra hệ thức (\*) với các giá trị x < n và (x,n) = 1

(\*)

Ta cài đặt hàm Exp(x, m, n) = xm mod n bằng kỹ thuật chia để trị.

Ta có

# z = x^m mod n

def Exp(x, m, n):

z = 1

while m > 0:

if m % 2: z = (z\*x) % n # m lẻ

m //= 2

x = (x\*x) % n

return z

def IsCarmichael1(n):

for x in range(2,n):

if Gcd(x,n) == 1:

if Exp(x,n,n) != x: return False

return True

Ghi nhớ

* Số Carmichael là hợp số lẻ và số Carmichael nhỏ nhất là 561.

Độ phức tạp

Phương án 1 gồm 2 pha. Pha 1 gọi thuật toán Sieve với độ phức tạp n log log n. Pha 2 duyệt k hợp số lẻ x trong khoảng 3..n. Với mỗi số x phải gọi hàm Gcd x lần và tính Exp. Mỗi lần tính hàm Gcd cần khoảng 5len(x) phép tính. Vậy Pha 2 cần cỡ km phép tính, trong đó m là độ dài trung bình của các số nguyên. Tổng hợp lại, độ phức tạp có cỡ max(n log log n, km).

Phương án 2

Định lý sau đây của A. Korselt giúp ta xác định các số Carmichael.

***Định lý Korselt*** *Số lẻ n ≥ 3 là số Carmichael khi và chỉ khi n là hợp số và với mọi ước nguyên tố p của n ta có*

* *p2 không là ước của n*
* *p-1 là ước của n-1*

Nói cách khác định lý Korrselt cho ta biết số lẻ n ≥ 3 là số Carmichael khi và chỉ khi dạng phân tích ra thừa số nguyên tố của n có trên một thừa số nguyên tố và mỗi thừa số nguyên tố p đều có bậc 1 và p-1 là ước của n-1.

Ví dụ

561 = 3⋅11⋅7; 2, 10 và 6 đều là ước của 560.

1105 = 5⋅13⋅17; 4, 12 và 16 đều là ước của 1104.

Theo định lý Korselt ta duyệt lần lượt các ước nguyên tố p của n và kiểm tra các điều kiện

* p có bậc 1
* (n-1) mod (p-1) = 0

Độ phức tạp

Phương án 2 tránh được hàm Sieve nên việc kiểm tra mỗi số lẻ có độ phức tạp tương đương với thuật toán phân tích ra thừa số nguyên tố.

Chương trình

# Carmichael numbers

from time import time

MN = 40000

def Go():

if input(' ? ') == '.': exit(0)

# Các số nguyên tố

def ByteSieve(n):

b = bytearray([1]\*n)

b[0] = b[1] = 0

for i in range(4,n,2): b[i] = 0

for i in range(3, int(n \*\* 0.5) + 1, 2):

if b[i]:

for j in range(i\*i, n, i): b[j] = 0

return b

def Gcd(a, b):

return a if b == 0 else Gcd(b, a % b)

# z = x^m mod n

def Exp(x, m, n):

z = 1

while m > 0:

if m % 2: z = (z\*x) % n # m lẻ

m //= 2

x = (x\*x) % n

return z

def IsCarmichael1(n):

for x in range(2,n):

if Gcd(x,n) == 1:

if Exp(x,n,n) != x:

return False

return True

def Run1():

b = ByteSieve(MN)

c = 0 # counter

for x in range(3, MN, 2):

if not b[x]: # x là hợp số lẻ

if IsCarmichael1(x):

c += 1

print(x)

print('Total:',c)

def IsCarmichael2(n):

n1 = n-1

count = 0 # đếm các thừa số nguyên tố

for p in range(3, int(n\*\*0.5)+1, 2):

if (n % p) == 0: # p là thừa số nguyên tố

n //= p

if n % p == 0: return False # p có bậc > 1

if n1 % (p-1) != 0: return False # n-1 phải chia hết p-1

count += 1

# Thừa số nguyên tố cuối cùng ?

if n > 1:

if n1 % (n-1) != 0: return False

count += 1

return count > 1

def Run2():

c = 0 # counter

for x in range(3, MN, 2): # xet cac so le

if IsCarmichael2(x):

c += 1

print(x)

print('Total:',c)

# APPLICATION

t = time()

print('Phương án 1')

Run1()

print(time()-t)

t = time()

print('Phương án 2')

Run2()

print(time()-t)

print(' T h e E n d')

Output

Phương án 1

561

1105

1729

2465

2821

6601

8911

10585

15841

29341

Total: 10

2.5884058475494385

Phương án 2

561

1105

1729

2465

2821

6601

8911

10585

15841

29341

Total: 10

0.27483081817626953

T h e E n d

## Maximum Prime

Thuật toán

* Dùng thuật toán Sàng Eratosthenes, Sieve tìm các số nguyên tố trong khoảng 2..1M. Chuyển các số này sang string và ghi vào danh sách p theo chiều giảm dần.
* Với mỗi string x đọc từ input file PRIMES.INP duyệt tìm số nguyên tổ đầu tiên (tính từ lớn trở xuống) trong p xem có phải là một đoạn của x ?

str x có trong str y ?

y.index(x) cho ra vị trí đầu tiên i nơi x bắt đầu xuất hiện trong y.

Ví dụ

'912345'.index('123') = 1

'91243'.index('123') báo lỗi : 'substring not found'

Bạn chỉ nên gọi hàm index khi biết chắc rằng str x có trong str y.

Chương trình

# Maximum Prime

from time import time

MN = 1000000

def Go():

if input(' ? ') == '.':

exit(0)

# Các số nguyên tố

def ByteSieve(n):

b = bytearray([1]\*n)

b[0] = b[1] = 0

for i in range(4,n,2): b[i] = 0

for i in range(3, int(n \*\* 0.5) + 1, 2):

if b[i]:

for j in range(i\*i, n, i): b[j] = 0

return b

# Tìm số nguyên tố p[i] trong x

def Match(p, x):

for i in range(len(p)):

if not (p[i] in x): continue

else: return i

return -1

def ReadInput():

with open("MAXPRIM.INP") as f:

d = f.read().split()

d.pop() # bo phan tu 0 cuoi day

return d

def Run():

# Phase 1:

b = ByteSieve(MN) # Sinh cac so nguyen to < 1M

# tạo dãy str các số nguyên tố giảm dần

p = [str(i) for i in range(MN-1,0,-1) if b[i]]

print('Total ', len(p), ' primes.')

# Phase 2

for x in ReadInput():

i = Match(p, x)

if i > -1:

print(x, 'found', p[i])

# APPLICATION

t = time()

Run()

print(time()-t)

print(' T h e E n d')

Output

Total 78498 primes.

11245 found 11

91321150448 found 1321

1226406 found 2

372441465 found 2441

2001368741189 found 41189

505099997967743 found 999979

0.8724653720855713

T h e E n d

Độ phức tạp

Sieve cần n log log n phép tính. Duyệt m số nguyên tố gọi hàm find m lần cho mỗi số với chiều dài trung bình k sẽ cần mk phép so sánh. Tổng hợp lại thuật toán cần max(n log log n, mk) phép toán cho mỗi số x.