## Divide And Conquer

Chia để trị là phương pháp thiết kế thuật toán dựa trên ý tưởng chia miền dữ liệu cần xử lý thành nhiều miền nhỏ, xử lý từng miền, sau đó tổng hợp kết quả.

Một câu hỏi quan trọng liên quan đến phương thức chia là: *Chia miền dữ liệu thành bao nhiêu phần ?*

Tuỳ theo nội dung của bài toán mà có các phương thức chia khác nhau. Dưới đây là một số ý tưởng chỉ đạo cho cho các phương thức chia:

* Nên chia thành các miền dữ liệu không giao nhau để khi tổng hợp được dễ hơn.
* Chia dữ liệu đủ nhỏ để bài toán vận dụng cho dữ liệu đó trở thành tầm thường, tức là trở thành tối giản.

## Bài toán tìm kiếm

Cho tập đối tượng S và một phần tử *x*. Hãy cho biết *x* ∈ S?

Tuỳ theo cấu trúc của tập S mà ta có các thuật toán tìm kiếm khác nhau.

Nếu S là tập hợp không được sắp thứ tự thì ta phải duyệt tuần tự từng phần tử trong S gặp phần tử đầu tiên S(i) = x ta return i, ngược lại, sau khi duyệt hết S mà không tìm được giá trị x trong S ta retturn -1. Thuật toán này gọi là *tìm kiếm tuần tự* và có độ phức tạp n = len(S).

Nếu S là mảng được sắp thì ta có thể thực hiện *tìm kiếm nhị phân* với độ phức tạp log(len(S)).

#### Tìm kiếm tuần tự

### Substring

*Xâu x được gọi là xâu con của xâu y nếu sau khi xoá đi một số ký tự trong y thì thu được x.*

*Hãy viết hàm Substr(string x, string y) cho ra số ký tự cần xoá khỏi y để thu được x. Nếu không thể xoá được, hàm cho ra giá trị -1.*

Ví dụ

Substr("abc","a12b34bb5c6") = 8

Substr("abc","abc") = 0

Substr("abc","a1cb") = -1

#### Algorithm

Gọi lenx và leny lần lượt là chiều dái của các string x và y. Xét ba trường hợp

* lenx > leny: bài toán vô nghiệm, return -1
* lenx = leny: return 0 if x == y else -1
* lenx < leny: gọi x[i] là ký tự đang chờ xét của x. Duyệt lần lượt các ký tự j của y. Nếu y[j] = x[i] thì ta chuyển qua xét ký tự chờ tiếp theo của x là x[i+1]. Nếu x hết ký tự chờ là thành công. Số ký tự cần xoá sẽ là leny-lenx.

#### Độ phức tạp

Cỡ len(y) phép so sánh.

#### Program

# Substring

def Go(msg = ' ? '):

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

# cần xóa k kí tự của y để thu được x

def Substr(x, y):

print(' x:', x, ' y:',y)

lenx, leny = len(x), len(y)

if lenx > leny: return -1

if lenx == leny: return 0 if x == y else -1

# lenx < leny

i = 0

for c in y:

if c == x[i]:

i += 1

if i == lenx: return leny - lenx

return -1

def Run():

print(' Result:', Substr("abc","a12b34bb5c6")) # 8

print(' Result:', Substr("abc","abc")) # 0

print(' Result:', Substr("abc","a1cb3")) # -1

# APPLICATION

Run()

print(' T h e E n d')

#### Output

x: abc y: a12b34bb5c6

Result: 8

x: abc y: abc

Result: 0

x: abc y: a1cb3

Result: -1

T h e E n d

#### Tìm kiếm nhị phân

Tìm kiếm nhị phân được vận dụng trong tập dữ liệu được sắp S. Muốn tìm phần tử x trong S, ta chia S thành hai phần A và B, S = A|B rồi kiểm tra xem x có thể thuộc phần nào. Nếu x ∈ A hoặc x ∈ B ta lại chia tiếp nửa này thành hai phần và lặp lại bước trước cho đến khi chỉ còn lại một phần tử. Do mỗi lần chia 2 ta bỏ đi một nữa miền tìm kiếm nên độ phức tạp sẽ là cỡ log len(S).

### Search(a,x)

*Tìm vị trí xuất hiện đầu tirn của số nguyên x trong dãy số nguyên được sắp tăng a*

# Tìm nhị phân

def BinSearch(a, d, c, x):

while d < c:

m = (d + c) // 2 # diem giua

if a[m] < x: d = m + 1

else: c = m

return d if a[d] == x else -1

### RSearch(a,x) (đệ quy)

*Tìm vị trí xuất hiện đầu tirn của số nguyên x trong dãy số nguyên được sắp tăng a*

# Đệ quy

def RSearch(a, d, c, x):

if d == c:

return d if a[d] == x else -1

# d < c

m = (d + c) // 2

return RSearch(a, m+1, c, x) if a[m] < x else RSearch(a, d, m, x)

### LastSearch(a,x)

*Tìm vị trí xuất hiện cuối cùng của số nguyên x trong dãy số nguyên được sắp tăng a*

# Tim chi so i phai nhat a[i] = x

def LastSearch(a, d, c, x):

while d < c:

m = (d+c+1) // 2 # diem giua

if a[m] <= x: d = m

else: c = m-1

return d if a[d] == x else -1

RLastSearch(a,x) (đệ quy)

*Tìm vị trí xuất hiện cuối cùng của số nguyên x trong dãy số nguyên được sắp tăng a*

# Đệ quy: Tìm chỉ số phải nhất i thỏa a[i] = x

def RLastSearch(a, d, c, x):

if d == c: return d if a[d] == x else -1

# d < c

m = (d+c+1) // 2 # diem giua

return RLastSearch(a,m,c,x) if a[m] <= x else RLastSearch(a,d,m-1,x)

Demo

# BinSearch

def Go(msg = ' ? '):

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

# Tìm nhị phân

def Search(a, d, c, x):

while d < c:

m = (d + c) // 2 # diem giua

if a[m] < x: d = m + 1

else: c = m

return d if a[d] == x else -1

# Đệ quy

def RSearch(a, d, c, x):

if d == c: return d if a[d] == x else -1

# d < c

m = (d + c) // 2

return RSearch(a, m+1, c, x) if a[m] < x else RSearch(a, d, m, x)

# Tim chi so i phai nhat a[i] = x

def LastSearch(a, d, c, x):

while d < c:

m = (d+c+1) // 2 # diem giua

if a[m] <= x: d = m

else: c = m-1

return d if a[d] == x else -1

# Đệ quy: Tìm chỉ số phải nhất i thỏa a[i] = x

def RLastSearch(a, d, c, x):

if d == c: return d if a[d] == x else -1

# d < c

m = (d+c+1) // 2 # diem giua

return RLastSearch(a,m,c,x) if a[m] <= x else RLastSearch(a,d,m-1,x)

def Demo():

a = [8, 10, 25, 10, 50, 25, 3, 10, 20, 5]

a.sort()

d, c = 0, len(a)-1

while True:

print(' a: ', a)

x = int(input(' (Get x < 0 to stop) x = '))

if x < 0: break

print(' Found the first appearance ', Search(a, d, c, x))

print(' Found the last appearance ', RLastSearch(a, d, c, x))

# APPLICATION

Demo()

print(' T h e E n d')

### Secret numbers(d,c)

*Với ít nhất các câu hỏi Y/N đoán ra số x trong khoảng d..c*

# BinSearch

def Go(msg = ' ? '):

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

# Tìm nhị phân

def SecretNumber(d, c):

print('Select a number between',d,'..',c)

input('Now, please, press RETURN to begin: ')

x = SN(d, c)

print('Your number is', x)

def SN(d, c):

while d < c:

m = (d + c) // 2 # diem giua

while True:

ans = input('Your number > '+str(m)+'? [Y/N]: ')

a0 = ans.capitalize()[0]

if a0 in 'YN': break

if a0 == 'Y': d = m + 1

else: c = m

return d

# APPLICATION

SecretNumber(1, 10)

print(' T h e E n d')

### Birthday

*Với không quá 9 câu hỏi Y/N có thể đoán được simh nhật của một người.*

# BinSearch

def Go(msg = ' ? '):

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

# Tìm nhị phân

def Birthday():

day = SN(1,31,' \* Your born day > ')

month = SN(1,12, ' \*\* Your born month > ')

print('Your birthday is ', day, '-', month)

def SN(d, c, ask):

while d < c:

m = (d + c) // 2 # diem giua

while True:

ans = input(ask+str(m)+'? [Y/N]: ')

a0 = ans.capitalize()[0]

if a0 in 'YN': break

if a0 == 'Y': d = m + 1

else: c = m

return d

# APPLICATION

Birthday()

print(' T h e E n d')

### Cubic Numbers

*Số bậc ba là số x có dạng với a là số một nguyên dương.*

*Cho một số nguyên dương N gồm tối đa 10 chữ số.*

*Hãy xóa bớt một số chữ số của N để thu được số bậc ba lớn nhất.*

Ví dụ

|  |  |  |
| --- | --- | --- |
| CUBIC.INP | Results | Giải thích |
| 7  4125  976  9413192  10007000  238  127731561  438698939 | 125  -1  941192  1000000  8  1771561  3869893 | 7 test  125 = 5^3  vô nghiệm  941192 = 98^3  1000000 = 100^3  8 = 2^3  1771561 = 121^3  3869893 = 157^3 |

Algorithm

Giả sử bạn đã có số N. Việc đầu tiên là bạn đánh giá bản thân N gần sát với số bậc ba v nào nhất.

Ta dùng ký hiệu N ≈ v.

Ví dụ

4125 ≈ 4913 = 17^3

50653 ≈ 50653 = 37^3

Nếu N = v thì bạn cho ngay kết quả. Ngược lại, khi N < v bạn duyệt các số bậc ba x từ v trở xuống

và so khớp số x với N. Nếu mọi chữ số của x đều lần lượt xuất hiện trong N thì x chính là số cần tìm.

Ví dụ

Bạn đã xác định được

N = 4125 ≈ 4913 = 17^3

Bạn lần lượt duyệt các số từ 17^3 trở xuống. Đó là

16^3 = 4096. So khớp 4096 với 4125: thất bại

15^3 = 3375. So khớp 3375 với 4125: thất bại

14^3 = 2744. So khớp 2744 với 4125: thất bại

...

5^3 = 125. So khớp 125 với 4125: khớp

Tốt nhất, bạn nên cài đặt hàm so khớp dưới dạng string. Cho hai số dạng string sy và sx.

Nhiệm vụ của ta là xét xem mọi chữ số của sy có lần lượt xuất hiện trong sx hay không ?

Ví dụ sau đây minh họa thuật toán.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| sy | **5** | **0** | **6** | **5** | **3** |  |  |  |  |
| sx | 2 | **5** | 1 | **0** | 9 | **6** | **5** | 8 | **3** |
| chỉ số | *0* | *1* | *2* | *3* | *4* | *5* | *6* | *7* | *8* |

# So khop cac chu so cua so bac 3 string sy trong string sx

def Match(sy, sx):

if len(sy) > len(sx): return False

if len(sy) == len(sx): return sy == sx

i = 0

for d in sx:

if d == sy[i]:

i += 1

if i == len(sy): return True

return False

Việc còn lại là khởi trị hai mảng:

Mảng num3 chứa các số bậc 3 không quá 10 chữ số.

Mảng snum3 chứa các số bậc 3 tương ứng dưới dạng string.

Vì 2155^3 là số nhỏ nhất có 11 chữ số, nên ta đặt giới hạn cho các số sinh ra số bậc ba

là MN = 2155.

num3 = [x\*\*3 for x in range(MN)]

snum3 = list(map(str, num3))

Khi tìm số sát nhất với số bậc ba ta dùng phương pháp tìm kiếm nhị phân trong mảng num3.

# Tim vi tri dau tien d: num3[d] >= x

# Phuong phap chia de tri log(MN)

def Search(x):

d, c = 1, len(num3) - 1

while d < c:

m = (d + c) // 2

if num3[m] < x: d = m + 1

else: c = m

return d

Program

# Cubic Numberd

MN = 2155

def Go(msg = ' ? '):

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

# Tim vi tri dau tien d: num3[d] >= x

# Phuong phap chia de tri log(MN)

def Search(x):

d, c = 1, len(num3) - 1

while d < c:

m = (d + c) // 2

if num3[m] < x: d = m + 1

else: c = m

return d

# So khop cac chu so cua so bac 3 string sy trong string sx

def Match(sy, sx):

if len(sy) > len(sx): return False

if len(sy) == len(sx): return sy == sx

i = 0

for d in sx:

if d == sy[i]:

i += 1

if i == len(sy): return True

return False

def Run():

global num3, snum3

# Init()

num3 = [x\*\*3 for x in range(MN)]

snum3 = list(map(str, num3))

with open("Cubic.inp") as f:

n = int(f.readline()) # so test

print(n, ' tests.')

for i in range(n):

sx = f.readline().replace('\n', '') # Test i

x = int(sx) # x: dang num, sx: dang str

r = -1

pos = Search(x) # vi tri dau tien num3[pos] >= x

print(' Test', i + 1, '.', x, end=': ')

for sy in snum3[pos::-1]: # duyet nguoc day snum3

if Match(sy, sx): # neu so khop

r = snum3.index(sy)

break

if r > 0: print(sy + ' = ' + str(r) + '^3')

else: print(r)

# APPLICATION

Run()

print(' T h e E n d.')

Result

7 tests

Test 1 . 4125: 125 = 5^3

Test 2 . 976: -1

Test 3 . 9413192: 941192 = 98^3

Test 4 . 10007000: 1000000 = 100^3

Test 5 . 238: 8 = 2^3

Test 6 . 127731561: 1771561 = 121^3

Test 7 . 438698939: 3869893 = 157^3

### 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 |  |  |

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.

Program

# Maximum Prime

from time import time

MN = 1000000

FN = "MAXPRIME.INP"

def Go(msg = ' ? '):

if input(msg) == '.': 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(FN) 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.

### Các lũy thừa tự thân (Self powers)

Thuật toán

Các số dạng được gọi là các *lũy thừa tự thân.* *k* chữ số cuối của số nguyên dương *x* là số Trong bài này *k* = 10. Do đó ta cần tính toán theo *phép chia dư* *m* (modulo *m*) với các số nguyên 64 bits, trong đó *m* = = 10000000000 (10G = mười tỷ).

Các quy tắc đơn giản dưới đây được vận dụng để tính toán:

(*a* + *b*) mod *m* = ((*a*

mod *m*) + (*b* mod *m*)) mod *m*

(*a* × *b*) mod *m* = ((*a* mod *m*) × (*b* mod *m*)) mod *m*

*ak* mod *m* = (*a* mod *m*)*k* mod *m*

*m* = 10G.

Tóm lại, nếu bạn chỉ quan tâm đến mười chữ số cuối cùng của kết quả thì mọi phép tính cộng, trừ và nhân trung gian (không kể phép chia) đều phải được tính toán theo mdulo *m* = = 10G.

Với *m* = 10G cho trước, ta cần triển khai các hàm sau đây:

Plus(a, b) = (a + b) mod m,

Mult(a, b) = (a × b) mod m, và

Exp(a, k) = ak mod m.

với yêu cầu là các hàm trên không được sinh ra các kết quả tính toán trung gian vượt quá giá trị *m.*

Để ý rằng mặc dù 0 ≤ *a, b* < *m*, nhưng có thể *a*+*b* ≥ *m*, nghĩa là tổng *a*+*b* có thể gây ra hiện tượng tràn số. Do đó ta cần thay phép kiểm tra *a*+*b* ≥ *m* bằng phép kiểm tra *a* ≥ *m* − *b* và thay phép tính (*a* + *b*) mod *m* bằng phép tính (*a* − (*m* − *b*)) vì ta luôn luôn có 0 ≤ *a, b* < *m.*

|  |
| --- |
| Algorithm Plus |
| Input: integer a, b; 0 ≤ a, b < m  Output: (a + b) mod m  begin  # (a+b) ≥ m ⇔ a ≥ m-b  if a ≥ m-b then return a - (m - b) # a+b-m  else return a + b;  end if  end Plus |

Ta cũng có thể xen trực tiếp đoạn mã của hàm Plus vào chương trình nhằm giảm thời gian truyền tham số.

Nếu x là bội của 10 thì ta bỏ qua, vì khi x = y10k, thì xx sẽ có trên 10 chữ số 0 tận cùng, do đó xx mod 10G = 0.

Chương trình

# Self powers

# Answer: 9110846700

# Time = 1.2981953620910645 giay.

from time import time

m = 10000000000 # 10G

def Go(msg = ' ? '):

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

# (a + b) % m

def Plus(a,b):

return a-(m-b) if a >= m-b else a+b

# z = xy mod m

def Mult(x, y):

z = 0

while x:

if x & 1: z = Plus(z,y)

x >>= 1 # x = x // 2

y = Plus(y, y)

return z

# z = a^x mod m

def Exp(a, x):

z = 1

while x:

if x & 1: z = Mult(z,a) # z = a\*z % m

x >>= 1 # x = x // 2

a = Mult(a,a) # a = a^2 % m

return z

def SelfPower(n):

sum = 1 # 1^1

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

if x % 10: sum = Plus(sum, Exp(x,x))

print(' Result:', sum)

print(" T h e E n d")

# run

t = time()

SelfPower(1000)

print(' Time = ', time() - t, ' giay.')

print('T h e E n d')

Output

Result: 9110846700

T h e E n d