**LỜI GIẢI CHI TIẾT MỘT SỐ BÀI TẬP TRONG CODE PTIT PYTHON**

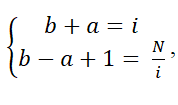
**PY01048. TỔNG LIÊN TIẾP**

Ta đặt tổng từ a đến b là N. Từ a đến b thì có (b – a + 1) số hạng. Ta dễ dàng tính được tổng theo công thức (Số cuối + Số đầu) \* Số số hạng /2. Hay ta có , 🡪 (b + a)(b – a + 1) = 2N

N là số tự nhiên nên 2 thừa số cấu thành nên tích kia cũng phải là số tự nhiên, nói cách khác thì (b + a) và (b – a + 1) phải là ước của 2N. Do đó, ta nhân N với 2 đã. Mà như ta đã biết, nếu i là ước của N thì N/i cũng chính là 1 ước của N. Tức là, (b + a) và (b – a + 1) một thằng là i, thằng kia là N/i.

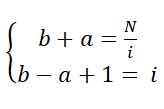
Hơn nữa, để xét hết tất cả các cặp ước thì chỉ cần xét các ước bé i(**tức là đến mà thôi**), lúc này sẽ tự sinh ra ước to N/i

Ta có 2 trường hợp sau:

a.  , suy ra b = (i + – 1)

Nếu b không nguyên tức là không tồn tại b (Đề bảo nguyên dương mà), hay (i + – 1)không chia hết cho 2, thì không tồn tại a. Khỏi xét

Nếu tồn tại b, ta dễ suy ra a = i – b = i - + . Mà như phương án xét đến căn n ban đầu thì i < N/i, suy ra ½(i - ) < 0, hay a < 1/2 . Do a nguyên dương nên không tồn tại a. Vậy không cần giải trường hợp này

b.  , suy ra b = (i + – 1)

Nếu b không nguyên tức là không tồn tại b (Đề bảo nguyên dương mà), hay (i + – 1)không chia hết cho 2, thì không tồn tại a. Khỏi xét

Nếu tồn tại b, ta dễ suy ra a = – b = - + . Trường hợp này vẫn có thể tồn tại a. Tuy nhiên, cần chú ý rằng đề bài yêu cầu a khác b. Nếu i = 1, lắp vào hệ phương trình này, ta sẽ suy ra được b + a = N, b – a = 0. Suy ra b = a = N/2. Không thỏa mãn

**Chốt lại, cách giải như sau:**

Nhân N với 2, vì ta phải xét ước của 2N mà

Duyệt i từ 2 đến **.** Xét theo trường hợp thứ 2 đã phân tích ở trên thì:

Nếu N chia hết cho i: b = (i + – 1)

Nếu b nguyên thì thực hiện tiếp bước sau, ngược lại quay lại xét i khác:

a = – b = - +

Đến đây, để cho chắc ta kiểm tra a, b đều >=1 và a < b là tăng số cặp lên

**PY01064 (Đệ quy). KÝ TỰ THỨ K**

(1): A

(2) = (1) + B + (1) = A**B**A

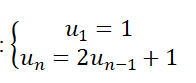
(3) = (2) + C + (2) = ABA**C**ABA

(4) = (3) + D + (3) = ABACABA**D**ABACABA

v.v….

**Từ các mẫu ví dụ, ta rút ra một số nhận xét sau:** Mã các ký tự A = 1, B = 2, C = 3, …

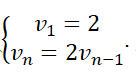
1. Ký tự thứ nhất trong mọi xâu luôn là chữ A
2. Đặt là độ dài xâu thứ n. Ta dễ thấy rằng mỗi một xâu thứ n sẽ gồm 2 xâu thứ n – 1 và ký tự có mã là n.

Ta có thể xây dựng hệ thức truy hồi cho dãy số như sau: 

Việc rút ra công thức tổng quát có thể sử dụng phương pháp dự đoán trực quan rồi chứng minh bằng quy nạp toán học hoặc biến đổi trực tiếp. Ở đây sẽ biến đổi trực tiếp

🡪

Đặt , ta suy ra , và .

Ta có dãy mới sau: . Dễ thấy rằng, đây là cấp số nhân với số hạng đầu là 2, công bội là 2. Do đó, công thức tổng quát của = 2.. Thay ngược trở lại công thức: . Ta được

Phân tích tiếp: Dãy thứ n = (Dãy thứ n – 1) + 1 ký tự mới mã là n + (Dãy thứ n – 1)

Do dãy thứ n – 1 dài – 1 (Theo công thức ta đã chứng minh), nên cái ký tự mới được bổ sung sẽ là ký tự thứ trong xâu thứ n. Vậy ta sẽ lấy mốc này làm chuẩn để tìm kiếm đệ quy

Gọi find(n, k) là tìm ký tự thứ k trong xâu thứ n (Tìm mã ký tự của ký tự nằm ở vị trí k):

Các trường hợp cơ sở:

K = 1: Luôn trả về ký tự ‘A”

K = : Trả về số n (Đã chứng minh trên, ký tự ở vị trí chính giữa luôn có mã là n)

K < : find(n – 1, k): Tìm ký tự thứ k trong xâu thứ n - 1

K > : find(n – 1, k - ): Số thứ tự của xâu thứ 3 đã bị dịch lên 1 đoạn bằng độ dài xâu THÀNH PHẦN thứ nhất (Xâu thứ n – 1) cộng thêm một (1 ký tự) nên phải trừ bớt.

Ta xây dựng hàm đệ quy:

find(*n*, *k*):

    if *k* == 1: return 1

    res = 2 \*\* (*n* - 1)

    if *k* == res: return *n*

    elif *k* < res: return find(*n* - 1, *k*)

    else: return find(*n* - 1, *k* - res)

**PY02017 (Trong đây ^ là ký hiệu phép XOR, không phải số mũ). TẦN SUẤT LẺ**

Phép XOR có các tính chất sau: Khác nhau XOR ra 1, giống nhau XOR ra 0.

Phép XOR có tính chất kết hợp

Vì vậy, khi XOR toàn mảng, theo đề ra, tất cả các số đều có tần suất chẵn duy nhất 1 số có tần suất lẻ. Ta sẽ kết hợp cụm các số có tần suất chẵn XOR với nhau, lúc này sẽ ra 0, dư ra 1 số lẻ thì là đáp án

Ví dụ: Dãy có con 1 xuất hiện 2 lần, 2 xuất hiện 4 lần, 3 xuất hiện 1 lần

A =(1 ^ 1) ^ (2 ^ 2 ^ 2 ^ 2) ^ (3^ 3)^3 = 0 ^ 0 ^ 0 ^ 3 = 3

Đây chỉ là mặt lí giải, còn khi ta code thì ta chỉ cần XOR toàn mảng từ đầu đến cuối thôi

**PY020250. ĐOẠN LIÊN TIẾP NHỎ HƠN**

Nói chung, khi dãy cho 1e5 phần tử thì 2 for là TLE chắc. Dạng toán tìm phần tử bên phải/trái đầu tiên >/</<=/>= nhìn chung là sử dụng stack rồi. Python thì list cũng đóng vai trò như stack với hàm .pop()

Về cách nhớ code, lắp code như sau:

1. Chiều xét:

Phần tử đầu tiên bên **TRÁI:** Ta xét từ bên **TRÁI** mảng đổ đi. Tức là for (int I = 0;i<n). Ngược lại, nếu đề bảo phần tử đầu tiên bên phải thì ta xét phần tử bên PHẢI đầu tiên đổ về

1. Điều kiện pop

Ví dụ: Phần tử đầu tiên **lớn hơn**

Ta muốn đỉnh stack chính là thằng đầu tiên > thằng a[i] đang xét. Nên stack sẽ chỉ được lưu những thằng mà > a[i] đang xét. Hay cứ đỉnh stack <= thằng đang xét thì pop ra: while st and a[st[-1]]<=a[i]:st.pop()

Đề cho điều kiện tìm thằng đầu tiên <, >=, <= thì tự suy ra dấu như đã phân tích ở trên.

Tùy vào yêu cầu bài toán mà stack ta sẽ lưu vị trí hoặc giá trị

Có thể áp dụng tư tưởng này cho bài PYKT12019.

**PY02067. DÃY SỐ TƯƠNG THÍCH:**

**Bài này thì giải toán khá nặng, cú pháp thì không có mấy**

Theo đề ra, ta thấy dãy B là dãy không âm, và , p nguyên và đẳng thức thỏa mãn với mọi i (0<=i<N). Trong này, [x] là phần nguyên của x

***Bước 1: Xác định điều kiện của p để với mọi* a[i] *thì luôn tồn tại* b[i].**

1. ***Giải bất phương trình ẩn* b[i] *với tham số* p**

Ta giải phương trình: 🡪 p , p nguyên

Với a[i] đã có, ta sẽ giải bất phương trình này theo p để tìm miền của b[i] theo p. Rồi từ miền đó suy ra điều kiện tồn tại của b[i]

Từ bất phương trình trên, suy ra:. Đặt

Ta có: b[i]>x(1) và b[i]<=y, b[i] nguyên

* Cận dưới

TH1: x không nguyên, ví dụ x = 6.5 thì b[i] >6.5, b nguyên 🡪 b[i] >= 7

TH2: x nguyên, VD x = 6, b[i] > 6, b nguyên thì b[i] >=7

Vậy chốt lại cận dưới là b[i] >=[x] + 1

* Cận trên:

TH1: y không nguyên, ví dụ y = 7.5 thì b[i] <= 7.5, b[i] nguyên 🡪 b[i] = 7

TH2: y nguyên, VD y = 7, b[i] <=7, b[i] nguyên 🡪 b[i] = 7

Chốt lại, cận dưới là b[i] <= [y]

Như vậy, ta có : [x] + 1 <= b[i] <= [y]

Vậy, ta có: . (1)

1. ***Tìm điều kiện của tham số p để tồn tại nghiệm* b[i]**

Dễ thấy bất phương trình (1) chỉ tồn tại nghiệm b[i] khi cận trên <= cận dưới. Vậy, với một số p, điều kiện để với mọi số a[i] luôn tồn tại b[i] là: .

***Bước 2: Sau khi đã xác định điều kiện của p để với dãy a luôn tồn tại dãy b, ta sẽ xác định xem p thế nào thì dãy b bé nhất. Và với mỗi p thì b thế nào là bé nhất***

Dễ thấy rằng, p càng to thì b[i] càng bé. Tuy nhiên, p không thể to vô hạn được.

Ta có đánh giá sau đây:

Nếu p > a[i] thì (như kiểu ), và hiển nhiên p + 1 > a[i] + 1 > a[i], nên .

Thay vào bất phương trình (1) ta có: 1 <= b[i]<=0. Vô lí, không có nghiệm

Do vậy, trong mảng a, p to nhất chỉ được phép = min(mảng A) (để số p <= a[i] với mọi i)

Và khi đã có số p to nhất thỏa mãn rồi, thì b[i] min sẽ là .

***Vậy, qua các phân tích trên, tóm lại cách làm như sau:***

1. ***Xác định điều kiện của số p để với số p đó, với mọi a[i] luôn tồn tại b[i]:***

def check(p):

for i in a:

if i//p < i//(p + 1) + 1: return 0

return 1

1. ***Tìm p to nhất thỏa mãn, với p tìm được thì tìm b[i] bé nhất:***

for i in range(s, 0, -1) :#Tìm số p max thỏa mãn. Tức là xét từng số i từ min mảng

if check(i) :#Số p max thỏa mãn

for j in range(n) : ans += a[j]//(i+1)+ 1

break

print(ans)