Bội số của K

Giải bài toán bằng phương pháp qui hoạch động. Đặt $f[i,j] = \pm u$ nếu như có thể thành lập biểu thức của i số đầu tiên để được một số chia cho k dư j $(0 \leq j < k)$ ở đây $f[i,j] = u$ nếu a_u là phần tử cuối cùng tham gia vào biểu thức với dấu dương, $f[i,j] = -u$ nếu a_u là phần tử cuối cùng tham gia vào biểu thức với dấu âm. Nếu không thể thành lập được số dư j thì $f[i,j] = 0$.

Mảng f có thể được tính từ dòng 1 đến dòng n dựa vào các giá trị $\neq 0$ của hàng trước đó với chú ý $f[0,0] = 1, f[0,j] = 0$

Nếu $f[n,0] = 0$ thì không thể thu được biểu thức chia hết cho k. Trường hợp ngược lại thì có thể và mảng f giúp ta tính được biểu thức cụ thể.

Giống nhau

- Subtask 1 (50%): Duyệt.
- Subtask 2 (50%): Dùng thuật toán Z.

Đàn bò hỗn loạn

Quy hoạch động trạng thái - Độ phức tạp O(2^N * N^2)

Gọi s là số nhị phân 16 bits, có x bits 1 (16 - x bits 0), với ý nghĩa nếu bit thứ i là 1 thì con bò thứ i được chọn để sắp xếp vào hàng.

=> s là trạng thái mà x con bò đã được chọn để sắp xếp vào hàng theo thứ tự hỗn loạn (không quan tâm đến vị trí các con bò được s/x như thế nào).

Gọi f[i][s] = số cách để xếp con bò thứ i vào cuối hàng(hoặc đầu, sao cũng được) theo thứ tự hỗn loạn với trạng thái s (tất nhiên bit thứ i của s phải được bật để sắp con bò i).

Gọi preS là trạng thái có được từ s bằng cách tắt bit thứ i (cho bit i = 0), có nghĩa là xếp bò xong trạng thái preS, tiếp theo xếp thêm con bò thứ i thì ta được trạng thái s (preS là trạng thái ngay trước s).

Xét lần lượt chỉ số của con bò thứ i với chỉ số j của các con bò trong trạng thái preS, nếu abs(a[i]-a[j]) > K (có nghĩa là xếp được với con bò thứ j), thì : **f[i][s] += f[j][preS]** (bò j và bò i lúc này đứng sau cùng của hàng và ghép thành một cặp với nhau, nên số cách xếp bò i thoả mãn cũng là số cách xếp bò j thoả mãn), nếu ngược lại abs(a[i] - a[j]) <= K, thì không xếp được => không cộng.

Kết quả là tổng f[i][S] với i = 0..N, S là cấu hình full bit 1 (đã chọn hết tất cả bò)

SỐ NGŨ HÀNH

Phương pháp quy hoạch động: Quy hoạch động kết hợp với xử lý số lớn

Một số chia hết cho 5 nếu và chỉ nếu các chữ số tận cùng của nó là 0 hoặc 5
Giả sử ta có xâu S, không quá lớn, ta lặp qua tất cả các phần tử của nó. Ta có thể làm như thế nào? Ta sẽ xây dựng một mảng là sol[].
Phương pháp quy hoạch động: Ta kí hiệu sol[i] là số cách để xóa các chữ số từ xâu S thỏa mãn số ngũ hành thu được có chữ số tận cùng ở vị trí thứ i.
Như vậy câu trả lời sẽ là sol[1]+sol[2]+….+sol[n]. Bây giờ ta tập trung vào việc tính toán sol[i]. Nếu s[i] (chữ số của xâu tương ứng với vị trí thứ i) là khác 0 hoặc 5, khi đó sol[i] sẽ bằng 0 (theo tính chất chia hết cho 5). Mặt khác, ta phải đặt ra một câu hỏi: "Có bao nhiêu cách để xóa các chữ số bên trái và bên phải vị trí i?" (Đây lại là bài toán đếm tổ hợp). Nếu xóa bên phải, ta chỉ có một cách đó là xóa tất cả các chữ số (nếu một chữ số từ bên phải vẫn còn, thì khi đó chữ số kết thúc không phải là ở vị trí thứ i). Bây giờ ta quan tâm tới việc xóa số ở bên trái: có các chữ số ở vị trí 1,2,…,i-1. Ta hoặc là xóa một chữ số hoặc giữ nó, dù thế nào ta vẫn nhận được một số là số ngũ hành. Vì vậy ở vị trí 1 có 2 cách (xóa hoặc giữ), ở vị trí 2 có 2 cách (xóa hoặc giữ),…. Và do đó ở vị trí thứ i-1 có 2 cách là (xóa hoặc giữ). Tiếp đến, ta áp dụng nguyên lý nhân trong toán học, ta thu được 2*2*2*…*2 (tất cả i-1 lần)=2mũ (i-1)
Để tính tổng nó ta làm như sau: Nếu S[i] bằng 0 hoặc 5 ta cộng vào biến đếm một lượng là 2^(i-1). Nếu khác thì ta không làm gì cả
Bài toán này đơn giản quy về bài toán tính A mũ B (là một số rất lớn) chia lấy dư cho một số?Bài toán này để tính nhanh được, ta dùng kỹ thuật Exponentiation by squaring
Phương pháp tính A mũ B nhanh có thể tham khảo tại :
Thuật toán bình phương và nhân:
https://vi.wikipedia.org/wiki/Thu%E1%BA%ADt_to%C3%A1n_b%C3%ACnh_ph%C6%B0%C6%A1ng_v%C3%A0_nh%C3%A2n
https://en.wikipedia.org/wiki/Exponentiation_by_squaring
http://stackoverflow.com/questions/5625431/efficient-way-to-compute-pq-exponentiation-where-q-is-an-integer
Độ phức tạp của thuật toán bình phương và nhân là O(logn)
Chú ý các tính chất sau:
(A * B) mod C = (A mod C * B mod C) mod C
Hay trong C++
(A*B)%C= ((A%C)* (B %C))%C
(A + B) mod C = (A mod C + B mod C) mod C
Hay trong c++
(A+B)%C= ((A%C)+ (B %C))%C
Phép chia:

(a/b) mod n = (a mod n) (b^{-1} mod n) mod n

Vậy phép chia này khi lập trình trong C++ sẽ làm như thế nào? Ta sẽ làm như sau:

Tình huống n là số nguyên tố, ta còn có định lý Ferma:

Định lý nhỏ của Fermat (hay định lý Fermat nhỏ - phân biệt với định lý Fermat lớn) khẳng định rằng nếu *p* là một số nguyên tố, thì với số nguyên *a* bất kỳ, $a^p - a$ sẽ chia hết cho *p*. Nghĩa là:

$$a^p \equiv a \pmod p$$

Một cách phát biểu khác của định lý như sau: nếu *p* là số nguyên tố và *a* là số nguyên nguyên tố cùng nhau với *p*, thì $a^{p-1} - 1$ sẽ chia hết cho *p*. Bằng ký hiệu đồng dư ta có:

$$a^{p-1} \equiv 1 \pmod p$$

Xem thêm về tính chất đồng dư thức tại :
https://vi.wikipedia.org/wiki/%C4%90%E1%BB%93ng_d%C6%B0

Nếu:

$$a_1 \equiv a_2 \pmod n$$

$$b_1 \equiv b_2 \pmod n$$

Thì $a_1 b_1 \equiv a_2 b_2 \pmod n$

Vậy nếu:

$$a^{p-1} \equiv 1 \pmod p$$

Thì

$$a^{p-2} = a^{p-1}.a^{-1} \equiv 1.a^{-1} = \frac{1}{a} \pmod p$$

Hay: $a^{p-2} \equiv \frac{1}{a} \pmod p$ (*)

Phần (*) và kinh nghiệm để tính a^{-1}

Việc cuối cùng là xử lý xâu s trong trường hợp lớn, tức là nhiều xâu giống nhau ghép lại với nhau, ta sẽ làm như thế nào? Câu trả lời xin để cho các em tự trả lời, ta phải nhân thêm với một lượng là bao nhiêu?

(gợi ý $\frac{(2^{k*l}-1)}{2^l-1}$ trong đó l là chiều dài của xâu input, còn k*l chính là chiều dài của xâu đã cho)

SHOPSIGN

Chú ý:

+ Phân biệt tấm ván và tấm biển.

+ Các tấm ván cắt ra làm tấm biển phải trong một đoạn liên tiếp các tấm ván từ i đến j. (1.1)

+ Vì không có thước nên muốn **Pirate** cắt được theo chiều ngang thì phải có một tấm ván làm mẫu có nghĩa là trong tấm biển cần tìm phải có ít nhất một tấm ván giữ nguyên không bị cắt. (1.2)

+ Để đạt được 60% đến 80% số test chỉ việc xét i là cạnh của tấm biển cần tìm sau đó tìm left nhỏ nhất <=i và right lớn nhất >=i thỏa mãn a[left..right]>=a[i] nếu right-left>=a[i] thì cập nhật a[i] với Max. Chú ý: Nếu a[i]<=Max thì ta không cần xét tấm ván i nữa.

Hướng dẫn:

Gọi left[i] và right[i] lần lượt là gần i nhất thỏa mãn:

+ left[i]<=i; right[i]>=i.

+ Min(a[left[i]..right[i]])>=a[i]).

+ a[left[i-1]]<a[i] và a[right[i]+1]>a[i].

Như vậy ta dễ dàng thấy nếu right[i]-left[i]+1>=a[i] thì ta sẽ cắt được tấm biển có cạnh là a[i] và chứa tấm ván i.

Bây giờ ta tính lần lượt left[i] và right[i] bằng Stack.

Ta có nhận xét: Nếu a[i-1]<a[i] thì suy ra left[i]:=i trái lại nếu a[i-1]>=a[i] thì left[i] luôn nhỏ hơn hoặc bằng left[i-1] vì trong đoạn left[i-1]..i-1 các phần tử luôn lớn hơn hoặc bằng a[i-1] mà a[i-1]>=a[i] nên ⇒ các phần tử trong đoạn left[i-1]..i luôn lớn hơn hoặc bằng a[i].

Tương tự ta có nhận xét trên đối với right.

Ta dùng Stack lưu lại left[i], left[left[i]], và i để thu hẹp phạm vi tìm kiếm left[i], thay vì phải duyệt từ i-1 đến khi nào gặp phần tử nhỏ hơn, thì với những phần tử có chiều cao lớn hơn hoặc bằng a[i] thì nếu a[i+1]<=a[i] thì các phần tử đó cũng thỏa mãn i+1 nên ta chỉ cần tìm tiếp từ left[i-1] trở xuống,và ta chỉ cần lưu i vào Stack để so sánh a[i+1] với a[i] thôi thay vì phải lưu đoạn từ left[i] đến i. Còn nếu a[i+1]<a[i] thì left[i] luôn bằng i.

Nếu Stack rỗng thì ta có luôn có left[i] := i và đẩy i vào Stack.

Nếu Stack khác rỗng thì ta làm như sau:

Khởi tạo y:=i;

Chừng nào Stack chưa rỗng và a[Stack[Top]]>=a[i] thì ⇒Pop(x); (x chính là Stack[Top]) ta có y=left[x].

Khi đó left[i]:=y; Đẩy i vào Stack. Ta thấy trong cả hai trường hợp i đều được đẩy vào Stack để phục vụ cho việc tìm left[i+1].

Tính right[i] tương tự như tính left[i] chỉ khác duyệt các phần tử theo thứ tự ngược lại.