АНАЛИЗ НА РЕШЕНИЕТО НА ЗАДАЧА МАЖОРАНТ

Решение за 10 точки

Update: Просто правим a[p]=q – O(1)

Query: Използваме стандартния алгоритъм за намиране на мажорант. Пазим текущ мажорант и колко пъти се среща. Обработваме ново число. Ако то е равно на мажоранта, увеличаваме броя срещания с 1. Иначе ги намаляваме с 1. Ако те станат отрицателно число, текущото число става мажорант и броят срещания става 1. Възможно е и подмасивът да няма мажорант, затова пробваме дали текущият мажорант се среща достатъчно пъти. Но така трябва да пазим колко пъти се среща всяко число. Ако пазим срещанията в map, постигаме сложност O(n*n*log(n)).

Общо: O(m*n^2*log(n))

За всяка от следващите подзадачи, първо компресираме числата.

Решение за 25 точки

Update: Както в предното, но след update компресираме. – O(n*log(n))

Query: След като компресираме числата, те са ≤n, затова можем да пазим срещанията в масив. Така сложността става O(n*n).

Общо: O(m*n*n)

Решение за 45 точки

Update: Както в предното – O(n*log(n))

Query: Нека разделим числата на 2 групи - леки и тежки. Леките се срещат ≤C пъти, а тежките- >C пъти в целия масив.

Леките числа могат да бъдат мажорант на масив с големина най-много 2*C, затова ги минаваме всички заедно, като използваме алгоритъма от предната подзадача.

За тежките отговаряме, като първо направим масив b, като b[i]=1, ако a[i] е равно на сегашното число x и b[i]=-1, иначе. Ако x е мажорант на някой подмасив [i, j], то ∑b[k]>0, i≤k≤j.

Първо правим префиксните суми на b. Тогава ∑b[k]=prefix[j]-prefix[i-1], i≤k≤j.

Нека прибавим n към всеки prefix[i], за да може 0≤prefix[i]≤2*n.

Така, ако фиксираме j, трябва да намерим колко от предните префикси са в интервала [0, prefix[j]-1].

Очевидно можем да поддържаме update и query на fenwick tree, но това има сложност O(n*log(n)).

Но можем да забележим, че |prefix[i]-prefix[i+1]|=1, тоест винаги местим интервала с 1. Тоест можем да пазим колко пъти се среща всеки prefix и да правим update и query за O(1). Общо получаваме сложност O(n).

За всички леки сложността е O(n*C), а за всяко тежко е O(n). Но има най-много n/C тежки, затова цялата сложност е O(n*(C + n/C)). Ако изберем C=$\sqrt{n}$, общо сложността е O(n*$\sqrt{n}$)

Общо: O(m*n*$\sqrt{n}$)

Решение за 65 точки

Update: Можем да осъзнаем, че правим премахване на 1 елемент и добавяне на 1 елемент. Тоест можем лесно да го направим за O(n). Ще трябва да правим и update на сегментно, описано по-долу.

Сложността на този update е същата, както на query-то.

Query: Тук използваме divide and conquer.

Нека сега да сме на интервал [l, r], av=(l+r)/2.

Divide: Пускаме dc (l, av) и dc (av+1, r).

Conquer (Merge): Сега остават само тези подмасиви с l≤ p ≤av, av+1≤ q ≤r.

(1) Ако x е мажорант в [p, q], то x е мажорант в [p, i] и/или [i+1, q]. Лесно можем да докажем с допускане на противното.
Тоест ако изберем i=av, то x е мажорант в [p, av] и/или [av+1, q].
(2) Нека p се мени от av до l. Тогава в редицата [мажорант [p, av]] има най-много log(av-l+1) различни числа. По индукция можем да покажем, че ако преди x има y различни мажоранта, когато x става мажорант за първи път, то x се среща поне (2 на степен y) пъти. От там следва и твърдението.

Нека приемем, че мажорантът е от описания тип и се среща в лявата част, като може и в дясната. Този мажорант може да приема едва log(av-l+1) стойности, затова след като го фиксираме, прилагаме броене, подобно на това за тежките числа от предната подзадача.

Аналогично решаваме и случая, когато мажорантът в дясната част.

Трябва обаче да внимаваме, защото мажорантът може да е и двете части и не трябва да го броим 2 пъти.

Нека с F(n) означим сложността на решението.

F(n)=2*F(n/2) + O(n*log(n))

Значи F(n)=O(n*log(n)*log(n))

Това обаче е бавно. Но все пак можем да използваме сегментно, за да не пресмятаме отново отговор, който знаем. Можем да докажем, че на всяко ниво (освен първото) имаме най-много 2 merge-а, както описаните по-горе. Тоест на query имаме най-много n*log(n)+2*n/2*log(n/2) +…+2*(n/2^i)*log(n/2^i)<log(n)*[n+2*(n/2+n/4+n/8+…)]<log(n)*[n+2*n]=3*n*log(n)

Общо: O(m*n*log(n))

Решение за 100 точки

Тук всъщност числата ни трябват само сортирани.

Update: Както в предното, просто актуализираме за O(n)

Query: Нека фиксираме сегашното число. Имаме на кои позиции се намира.

Да забележим, че често за някое число подмасивите, където е мажорант, са несвързани. Тоест те образуват няколко групи.

Тоест например 1-цата в 1 1 7 2 3 4 7 1 1. Ще наричаме това „линия на цепене“, защото няма подмасив с мажорант сегашното число(1) и включващ и числа от лявата, и от дясната страна.

Всъщност, за да може i-я(на позиция p[i]) и j-я(на позиция p[j]) индекс на сегашното число да образуват масив с мажорант това число, трябва

2*(j-i+1)>p[j]-p[i]+1

2*j-2*i+1>p[j]-p[i] ⇔ 2*j-2*i >= p[j]-p[i] ⇔ 2*j-p[j]>=2*i-p[i]

Нека F[i]=2*i-p[i]

За да може да има „линия на цепене“ между i и i+1, трябва F[p]>F[q] за всяко p<=i и q>i.

Тоест трябва MIN[F[1],F[2]…F[i]]>MAX[F[i+1],F[i+2]…F[k]], k-брой срещания на сегашното число

Всички линии лесно можем да намерим за O(k)

Нека разгледаме пример:

1 1 2 3 2 1 2 1 2 2 1, сегашното число е 1.

Имаме k=5, p=[1,2,6,8,11], f=[1,2,0,0,-1] и значи имаме 2 „линии за цепене“- 1 2 ||| 0 0 ||| -1

Когато обаче цепим трябва да вземем и различни от текущото числа от 2-те страни.Да си представим, че вече в нашия разглеждан подмасив няма „линии на цепене“. Нека сегашният

подмасив има дължина d. Очевидно можем да оставим само по d числа от двете страни, които ги носим, заедно със сегашния подмасив. Колко е най-много дължината на този подмасив? Оказва се, че е 5*d. Тоест можем да приложим алгоритъма за тежките числа от 3-та подзадача.

Как доказваме това твърдение?

Нека първо се отървем от 2-те страни, които носим (те са с дължина до d всяка). Така остава да докажем, че разстоянието между първото и последното срещане на сегашния елемент, е най-много 3*d. Нека смятаме за „единици“ тези, които са равни на сегашния елемент.

Нека p[1]=1 и значи F[1]=1, тоест ще докажем, че p[d]<=3*d. Точната оценка е p[d]<=max(2*d-1, 3*d-4), но и p[d]<=3*d и достатъчно за нашите нужди.

Да сцепим числата на максимални групи, равни на сегашния елемент. За всяка група [l(l-тата единица), r(r-тата единица)] е изпълнено, че F[l]+1=F[l+1], F[l+1]+1=F[l+2] … F[r-1]+1=F[r]. Ако [l, r] и [r+1, q] са 2 съседни групи, то F[r]>=F[r+1] ⇔ 2*j-p[j]>=2*(j+1)-p[j+1] ⇔ p[j+1]>= p[j]+2, което е вярно, защото p[j+1] и p[j] не са свързани, заради максималността си.

Нека да си представим граф с върхчета всяко цяло число.

Нека да свържем за всяка група [l,r], F[l] с F[l+1], F[l+1] с F[l+2] … F[r-1] с F[r]. За да може да няма

„линии на цепене“, трябва F[1] да е свързано с F[d]. Но този граф има < d ребра, тоест F[d]>F[1]-d

2*d-p[d]>1-d => p[d]<3*d-1.

Защо трябва F[1] да е свързано с F[d]?

Да забележим, че F[i]+1=F[i+1], или F[i]>=F[i+1] и ребрата са само от вида (m,m+1).

Нека F[1] и F[d] не са свързани. Нека реброто (p,p+1) го няма и това ребро е по пътя между F[1] и F[d]. Тоест до някакъв момент винаги е изпълнено F[i]>p, и нека j е първото с F[j]<=p. То не може да стигне до p+1, защото това изисква да има ребро (p,p+1). Значи F[l]<=p за всяко j<=l. Значи j-1 и j образуват „линия на цепене“, защото F[i]>p>=F[l] за i<j и j<=l. Противоречие.

Така сложността за едно query става ~O(n+5*n), защото за 5*d операции махаме d числа от разглеждане.

Общо получаваме O(m*n)

Лесно можем да видим, че имаме големи константи на последните 2 решения, които ги забавят, а от там и съответно по-ниските ограничения.

Автор: Мартин Копчев