Анализ на задача `unique`

Тагове: офлайн заявки, дърво на Фенуик, wavelet дърво, персистентно сегментно дърво, merge-sort дърво, динамично сегментно дърво, 2D структури от данни

Задачата е вдъхновена от сравнително известната задача за брой различни стойности в подмасив (някои биха казали учебникарска, но зависи какви учебници гледаме!). Оказва се, че ако искаме не само различните, но уникалните стойности в даден подмасив, подходът се усложнява. За съжаление, за цялата задача с поддържане на онлайн заявки, заявките за промяна се оказват много по-тежки от въпросите, затова техният брой е ограничен по условие. Така е по-трудно на различни бавни "чийт" решения да минават тестовете, защото повечето от тях могат много бързо (константно или за логаритъм) да извършват заявка за промяна, разбира се на цената на по-бавен отговор.

Решение на първа подзадача – 10 точки

По традиция започваме с подзадача за наивното решение. Достатъчно е за всяко изследване да обходим числата в подмасива и с *count* масив да открием тези, които се срещат точно веднъж. Съответно за всяка промяна е достатъчно да сменим числото в масива. Голямата константа на по-сложните решения позволяват това решение да се справя сравнително добре дори при зададените по-големи ограничения за N и Q тук.

Сложност: $O(QN)$.

Решение на втора подзадача – 15 точки

Тук започва поредицата подзадачи, в които няма заявки за промяна. Допълнително улеснение е, че можем да обработваме заявките офлайн. Това позволява да пригодим подхода, който се използва за брой различни числа, с малко допълнение. Нека сортираме заявките по нарастващ ляв край. Може да си мислим, че имаме показалка за левия край, която започва от индекс 0 и се движи до индекс $N-1$. Съответно когато показалката е със стойност l, ще намираме отговорите на изследванията с ляв край равен на l. За тази цел ще отбелязваме за числата от масива с индекси $[l, N-1]$ с $+1$, -1 и 0 съответно дали са първото срещане на такава стойност в $[l, N-1]$, дали са второто срещане на такава стойност и дали са трето или по-голям номер срещане. По този начин, ако просто сумираме стойностите в подмасива на изследване $[l, r]$, сумата ще е броя уникални стойности в него, защото ще имаме точно по 1 за всяка уникална стойност, а за повтарящите се ще имаме сума 0, защото второто срещане ще нулира сметката за стойността, а последващите срещания на стойността не се отразяват на сметката (защото са нули).

Така е достатъчно в *дърво на Фенуик* да поддържаме съответните маркирания на числата от масива, за да може бързо да намираме сумата за подмасива на изследване. Когато придвижваме левия край надясно, то трябва да разгледаме следващото и по-следващото срещане на стойността (ако има такова), които съответно ще станат с маркери $+1$ и -1 (от предишни -1 и 0). Авторът допълнително маркира и текущата стойност, от която се отместваме, с маркер 0, като това позволява отговорът на изследване да е сумата на префикса $[0, r]$ (а макар и ненужно тук, също и по-лесна имплементация за поддържане на заявки за промяна).

Сложност: $O((N+Q)\log N)$.

Решение на трета подзадача – 13 точки

Това е първата подзадача с допълнителното условие. То всъщност улеснява търсения отговор, защото в подмасивите за изследване винаги знаем, че всяка стойност се среща веднъж или два пъти. Това позволява отговорът на въпроса да стане доста подобен на отговора на въпроса за броя различни стойности. Нека за всяка стойност записваме индекса на предишното ѝ

срещана в масива. Така ако имаме изследване на подмасива $[l, r]$, то броят уникални стойности ще получим като от $(r - l + 1)$ (броя числа в подмасива) извадим два пъти броя стойности, за които предишното срещане е в рамките на подмасива. Това е вярно, понеже стойностите, които не са уникални, се срещат точно два пъти.

Така свеждаме подзадачата до намиране на броя числа в подмасив (съответните индекси на предишните срещания), които са по-големи или равни на фиксирана стойност (l). Има разнообразни подходи, които решават тази задача. Най-директният може би е с *merge-sort* дърво, но това ще оставим за по-нататък, когато е полезно за заявките за промяна. Друг подход с по-малко известната структура от данни (и по-малко тромава от *merge-sort* дървото) е с *wavelet дърво*, с което може да се запознаете например от тук. Тя позволява и по-лесно поддържане на промени, но за съжаление в тази задача промените не са подходящи, за да се приложи и за цялата задачата. Тук ще опишем подхода с *персистентно сегментно дърво*, който всъщност можем да мислим като обобщаване на идеята в първата подзадача за онлайн заявки.

Достатъчно е за заявки с фиксиран ляв край да маркираме с $+1$ второто срещане на всяка стойност, за да броим повторенията при заявките. Това може да стане по аналогичен начин (и с по-малко работа) на описаното в първа подзадача. Сега преди да отговаряме на заявките симулираме местенето на левия край от 0 до $N-1$ и записваме историята на сегментното дърво (в случая е по-удобно да използваме сегментно дърво вместо Фенуик). След това директно ще можем да намираме отговорът на изследване $[l, r]$ с една заявка върху версията при ляв край l. Персистентното сегментно дърво позволява да не пазим наивно всички сегментни дървета, а по-умен начин да пазим единствено променените върхове при няколкото промени, които се извършват с местенето на левия край. (за сметката на повече памет)

Сложност: $O((N + Q) \log N)$.

Решение на четвърта подзадача – 45 точки (=0+15+13+17)

Описанието на предишните подзадачи всъщност ни дава директно решение на тази подзадача, която е цялата задачата, но в която няма промени. Можем да приложим персистентното сегментно дърво точно за промените при симулацията на местенето на ляв край, която правим в първа подзадача. Тук има и доста по-сложни подходи, на които няма да се спираме, а и не са толкова поучителни.

Сложност: $O((N + Q) \log N)$.

Решение на пета подзадача – 28 точки (=0+0+13+0+15)

TBD...

Сложност: $O((N + Q) \log^2 N)$ или $O(N \log 2N + Q \log^2 N)$.

Пълно решение – 100 точки

TBD...

Сложност: $O((N + Q) \log^2 N)$.

Автор: Илиян Йорданов