Анализ

Интересното в тази задача е, че решението за 100 точки е много добре оптимизирана версия на решенията за 5 точки.

Преди да почнем с каквото и да е решение, ние трябва да се питаме следния въпрос – защо търсим именно простите числа? С малко разсъждения, може да стигнем до следните две разсъждения:

- Всяко съставно число има делител, различен от 1.
- Всяко просто число няма делител, различен от 1.

Така, горните две разсъждения ни *крещят* да намерим единицата в пермутацията, да я изключим от нея и да продължим да работим с останалите стойности. Така, един подход би отстранил единицата като прави следното:

- Поддържаме кандидат-единица $cand$ измежду първите x стойности в пермутацията, като ако единицата е измежду първите x стойности, $P_{cand} = 1$, иначе P_{cand} е произволна позиция измежду първите x позиции (*инвариант* на алгоритъма за намиране на единицата).
- За да намерим новата стойност на $cand$ за първите $x+1$ позиции, ако е изчислен за първите x позиции, ще използваме заявка $query(x+1, cand)$ и прилагаме $cand := x+1$, ако заявката върне `true`. Ако $P_{cand} = 1$, то заявката ще даде `false`, а ако $P_{x+1} = 1$, то $cand := x+1$, откъдето алгоритъма ни е верен (защото `инварианта` е изпълнен и има единица в пермутацията).

Това ще използва $N-1$ заявки за намиране на единицата, но ще ни направи живота много по-приятен. В тази задача имаме късмета прагът за брой заявки за 100 точки да е щедър, та това не би било проблем.

След като сме премахнали единицата, задачата ни се свежда до следното – да намерим кои са стойностите, които не ги дели никоя друга. Така може да ни хрумне следната идея – да *чистим* с определена стойност p_i, което ще е да проверим за всички останали стойности p_j, дали $p_i | p_j$, и ако това е изпълнено, да отбележим, че p_j не е просто число. Директното прилагане на чистене с всички елементи би ни дали $N(N-1)$ заявки, което е твърде много. Заради това може да забележим, че ако за дадено число p_j вече сме отбелязали, че не е просто, не е нужно впоследствие да проверяваме отново дали $p_i | p_j$ за друго p_i. От там идва и името на тази процедура – когато намерим стойност a кратна на тази, с която чистим, ние *почистваме* пермутацията от съставната стойност a.

Малко пооптимизирани версии на горната идея биха ни дали към 20 точки. Следващия въпрос е ключов за решението и правилния ни отговор би ни вдигнал значително броя точки, а именно – с кои стойности има смисъл да чистим? Ние знаем, че броя двойки числа (x, y), такива че $1 \le x \le y \le N$ и $x | y$ са $\lfloor \frac{N}{1} \rfloor + \lfloor \frac{N}{2} \rfloor + \cdots + \lfloor \frac{N}{N} \rfloor \approx N \sum_{i=1}^{N} \frac{1}{i} \approx N \int_1^N \frac{1}{x} dx = N \log N$, откъдето може да очакваме, че едно случайно избрано число би изчистило около $logN$ негови кратни (естествено, с голяма вариация – построете графика на N/x), откъдето бихме могли да очакваме, че ще ни трябват $\frac{N}{logN}$ на брой *чистения*, за да почистим пермутацията от съставни числа, което е скъпо. От тук следва логичния въпрос – как да намираме добри стойности, с които да чистим?

Един бърз начин да пробваме това е решението `heuristic_sort_emo.cpp`, което е по идея на Емил Инджев. С него може за всяко число да се опитаме да оценим колко добре би се справило с чистенето. Заради това може да изберем случайни двойки (i, j) и да проверим дали $p_i | p_j$. Ако това е изпълнено, то увеличаваме броя *ударени* негови кратни от него. За да дадем еднакакъв *шанс* всяко едно число да се справи добре на горния *тест*, една идея е за всяко i да проверим еднакаква бройка j-та (което в решението е равна на 20) дали са негови кратни. Така, ние може да сортираме числата в редицата спрямо това колко *добре* са се представили на *тестa*, т.е. ненарастващо по бройката на

намерени кратни p_j. След това може да чистим редицата в реда на сортировката. Това решение изкарва около 60 точки, като се прилага допълнителната оптимизация да се "изчистят" кратните p_j, намерени в началото на решението, т.е. по време на самия *тест*.

Натурално продължение на горната идея е да разсъждаваме именно за вида на числата, които биха се представили добре на самия *тест*. Така следва този извод – представете си, че бихме избрали да чистим с определена стойност x и знаем, че за x има друга стойност y, такава че $y|x$. Тогава, ако чистим с y ще почистим не по-лоша бройка числа от пермутацията. Повтаряйки този аргумент многократно (т.е. ако $y|x$ и $z|y$, е по-оптимално да чистим с z вместо с x), ние достигаме до извода, че е най-оптимално да чистим с прост делител на x. Така, ако успеем да намерим прост делител p на x, може директно да чистим с него. Сега следва следния въпрос – как да намерим прост делител на x? Върнете се към идеята за намиране на единицата в пермутацията. Ако я приложим във вида на пермутацията, след като сме премахнали единицата, като в началото $cand$ е произволно, ще намери прост делител на $cand$ – като единицата липсва, няма кой да измести прост делител, който се е появил като стойност на $cand$, а такъв би се появил. Така може да приложим следното – поддържаме два списъка, един списък A, в който пазим вече намерени прости числа и друг списък B, в който пазим кандидат-простите, т.е., множество от числа, из което се намират тези прости, които не са в A. Така, избираме произволно число x от A и намираме негов прост делител p чрез горната стратегия (като проверяваме из цялата редица). С този прост делител чистим всяко негово кратно от A и продължаваме с алгоритъма. Тъй като сме сигурни, че p е просто, ние може да го добавим в B.

Това продължава да е лоша стратегия, защото търсенето на прост делител е твърде скъпо. Тук идва оптимизацията, която е причината тази идея да има смисъл. Забележете, че алгоритъма ни винаги чисти с прости числа. Така, ако сме избрали случайно число x от тези в списъка A, то всички негови прости делители ще бъдат в A (защото x не е изчистено), откъдето може да търсим прост делител на x с идеята за намираме на единицата директно в списъка A. Това решение изкарва около $60 - 70$ точки.

Може да го оптимизираме по следния начин – вместо да чистим с един прост делител на x, да чистим с всичките му прости делители. Това може да го направим по следния начин – намираме всичките делители на x, които ще бъдат в A, след което ако x има d делителя, ние правим $O(d^2)$ проверки за да намерим кои измежду тях са прости (аргументът за (просто число) $\Leftrightarrow$ (няма число, което го дели) важи и когато за $p|x$ проверяваме дали $y|p$ за $y|x$), след което чистим с всеки един делител от тях. Това решение е около $75 - 80$ точки.

За пълен брой точки ни трябва едно последно наблюдение. Ние може да видим, че има огромна бройка прости числа, които не делят нищо, т.е. са $> \frac{N}{2}$ (които по Prime Number Theorem са около $\frac{N}{2lnN}$, което за $N = 10^4$ е равно на 560). Така, в даден момент A ще бъде изпълнено само с тези прости числа, откъдето няма да прави нищо продуктивно, защото ще избира произволно просто число, ще намира кои са простите му делители, различни от него, (и няма да има такива), след което със самото избрано x ще провери из цялото дали има негови кратни, и няма да има такива. Може да видите, че една оптимистична оценка на заявките, използвани в този момент от нашата стратегия е $\approx 560^2 = 313\,600$, което е много. За наше щастие, има много добър и лесен начин да спестим голяма част от тях. Тъй като съдържа всички прости, които не сме открили към момента, то моментът, в който A се състои само от прости числа, то големината на A ще бъде равна именно на бройката неоткрити прости, което е именно равно на $\pi(N) - |B|$ (защото B съдържа всички открити прости), където $\pi(N)$ е стандартното означение за броя прости от 1 до N, а $|B|$ е големината на B. Така, като дойде този момент може да прекъснем алгоритъма си и да преместим всички стойности от A в B и функцията ни да върне B. С това сме готови.

Автор: Борис Михов