	На пълното решение	На частичните решения
Тагове	Делимости Наблюдения Алгоритъм на Евклид Решето на Ератостен	–

Анализ

Подзадача №1

Както винаги оставих подзадача с тестовите примери за обратна връзка от системата.

Подзадача №2

За подзадачата съм предвидил изкувствено усложнена версия на решението за следващата подзадача. Нейната цел е да позволи на повече решения да изкарат точки.

Постигната сложност: $O(max\ lg\ lg\ max + n^4 log_2 max)$

Имплементация: `divide_20p.cpp`

Подзадача №3

Едно от първите неща, които състезател трябва да забележи в задачата е, че числото, което използваме за делител, е просто. Съответно реда на операциите няма значение, защото различните p-та не влияят едни на други. Формално може безпроблемно да разменим две съседни операции и да получим един и същи ефект върху редицата, с което може да пермутираме множество от операции по избран от нас начин. Това ни дава възможността да разгледаме задачата поотделно за всяко p, защото така може да изчистим редицата от делители кратни на 2, после от делители кратни на 3 и т.н.

Така нека разгледаме един и същ прост делител p, който присъства в редицата. Той трябва да раздели всяко едно a_i точно определен брой пъти – по-точно равен на броя на срещанията му в каноничното разлагане на a_i. Нека тази бройка да бъде v_i. Така задачата ни се превръща в следната – разглеждаме редица $v_1, v_2, \cdots, v_n$ и можем да изберем неин подмасив, в който $\forall v_i > 0$. Намаляваме стойността на всеки един елемент в него с 1 и целим след минимален брой операции в редицата всичките ѝ елементи да са 0. Интуитивно тази трансформация може да бъде разгледана като логаритмуване на редицата с основа p.

За тази подзадача може да се приложи относително интуитивен подход – винаги да избираме за операцията най-дългия интервал с $\forall v_i > 0$. Доказателството на оптималността на този избор се оставя на читателя с подсказката да се търси противоречие по допускане на противното.

Оказва се, че няма значение за подзадачата дали ще търсим интервала по наивния $O(n^2)$ начин или чрез показалки за $O(n)$, защото и двете решения се справят достатъчно добре за $n \leq 500$, но са твърде бавни за $n \leq 5000$. Авторова грешка е, че няма подзадача, която да ги разграничава, но по класирането може да се забележи, че такъв не би допринесъл за тогавашното състезание.

Забележете, че в редицата ще приложим до $(n \log_2(max\{a_i\}))$ деления. Това е вярно, защото при деление на едно число, ние го разделяме с $p \geq 2$, откъдето ще може да му приложим най-много $\log_2$ деления. Така долните граници за решенията са обосновани при крайното допускане, че всяко едно

деление ще включва точно 1 число (което, на практика, се достига, както е споменато по-горе).

Сложности: $O(n^3 \log_2(max\{a_i\}))$ и $O(n^2 \log_2(max\{a_i\}))$ Имплементация: `divide_40p.cpp`

Подзадача №4

Една добра идея, която може да му хрумне на състезател, че последователното, повтарящо се и изморяващо намиране на най-дълъг интервал с кратни на p може да се прави паралелно по цялата редица. Иначе казано, представете си редицата $5, 1, 5, 5, 5, 5, 5, 1, 5, 5$. Текущото решение ще избере средния интервал от петици и ще игнорира първия и последния. Вместо това, може паралелно да ги намерим всичките наведнъж и да приложим операция за всеки от тях. Как? Ами, относително просто. Вместо да приложим операцията върху най-дългия интервал, ние ще намалим всяко $v_i > 0$ с 1 и ще изчислим минималния нужен брой операции за постигане на това. Изчислението на техния брой е относително лесно при досещане, че можем да разбием редицата от v-та на максимални подмасиви от ненулеви елементи. Така примерна редица $1, 2, 0, 3, 0, 4, 5$ ще я разбием на подмасиви $[1, 2]; [3]; [4, 5]$ и при всеки ще приложим 1 операция. Оптималността на този алгоритъм е на база на фактът, че всеки един от споменатите подмасиви ще бъде все някога най-дългият, който щяхме да изберем в решенията за горната подзадача (заради "независимостта" на тези подмасиви помежду им).

В горните решения броят на операции за едно p може спокойно да достигнат $n \log_p(max\{a_i\})$, защото в пример с $\{v\} = \{\log_p(2 \cdot 10^6)\}, 0, \{\log_p(2 \cdot 10^6)\}, \cdots$ се постига тази оценка. В текущото решение много лесно се вижда, че на едно просто число не може да се отдели повече от $\log_p$ операции, защото горната граница на v_i е именно това, а всяко едно обхождане на редицата намалява всяко ненулеви v_i с 1.

Разликите между решението на текущата подзадача и тези на горната е, че оценката от $O(n^2 \log_2(max\{a_i\}))$ вече не е приложима, заради подобренията в него. Не считам, че има добър вид на сложността, но в конкретния пример за $n = 5000$ е видимо, че текущото решение се спрява много по-добре от горните.

Импелментация: `divide_60p.cpp`

Подзадача №5

Първо решение.

В това решение използваме наблюдението, че общия брой деления е малък – до $(n \log_2(max\{a_i\}))$. То ще е такова, че ще се пробва всяка една итерация през елемент да бъде за сметка на прост делител, който текущата операция ще премахне.

Един начин да се реши задачата е решението ни да си прави решенията още по-паралелно. Това може да го постигнем като комбинираме "търсенията" за максимален интервал от **Подзадача №3** от различните прости в едно по-голямо търсене. Решението на **Подзадача №4** ни подсказва, че няма нужда винаги да търсим истинския максимален интервал, а че ни стига да разгледаме максималният такъв, почващ от първото a_i, различно от 1.

Ако успеем да намерим максималният интервал, почващ от такова a_i, за което съществува просто p, за което да приложим операция, ние ще успеем да си решим задачата бързо, защото ще следваме принципа, описан по-горе.

Това може да стане именно като постепенно итерираме към елементите надясно от a_i и търсим първия момент когато не съществува такова просто p. За бърза проверка дали текущия интервал

спазва условията, ние може да поддържаме НОДа на елементите в него. Ако този НОД е различен от 1 съществува такова p, иначе не. Например при редица $1, 36, 48, 24, 35$, разглеждания интервал ще е $[36, 48, 24]$ с НОД$= 12$, като НОДа на $[36, 48, 24, 35]$ е 1.

Така може да разгледаме кои прости числа делят НОДа в момента преди да стане равен на 1 (и по-колко пъти) и да приложим операциите за съответните прости. Също би могло да разделим всяко едно число на този НОД и да видим колко прости операции участват в този неин *комбиниран* вид. И за двата подхода са ни нужни Решето на Ератостен за бързо намиране на простите делители.

Забележете, че малко се отделяме от оригиналната ни идея да итерираме само през елементи, върху които впоследствие ще приложим деление, защото елементът при който стигаме НОД$= 1$ не го делим на нищо. Същевременно за всяка операция имаме до 1 такъв, откъдето тъй като общия брой операции е до $n \log_2(max\{a_i\}$, то в най-лошият случай ще направим $2 \cdot n \log_2(max\{a_i\})$ стъпки, което пак е $(n \log_2(max\{a_i\}))$. Забележете, че в долната сложност на решението се включва, че сложността на Алгоритъма на Евклид е до $O(\log_2(max\{a_i\})$.

Постигната сложност: $O(n \log_2^2(max\{a_i\}))$

Имплементация: `divide_100p.cpp`

Алтернативен подход.

Може по друг начин да разгледаме паралелно операциите. За да преброим общия им брой, ние може да изчислим колко завършват във всеки един елемент a_i. Представете си една операция, която завършва в елемент $_i$, различен от крайния. Логично следва въпроса – защо не сме включили и a_{i+1} към операцията? Това би се случило единствено когато операцията е за такова p, което след изпълнението на операциите преди текущата, вече не дели a_{i+1}.

Ние всъщност знаем точният брой на операциите, в които участва един елемент a_i – за $a_i = \prod_{i=1}^{k} p_i^{\alpha_i} = p_1^{\alpha_1} p_2^{\alpha_2} \cdots p_k^{\alpha_k}$ ще е $\alpha_1 + \alpha_2 + \cdots \alpha_k$, т.е. за всеки един прост делител в каноничното разлагане на a_i броим по 1 операция.

Тогава, за да завършва една операция за определено p в a_i, то трябва p да се среща повече пъти в каноничното разлагане на a_i, спрямо a_{i+1}. По-точно, нека срещанията на p в каноничното разлагане на x да е $\nu_p(x)$. Тогава за това просто p ще завършват $\max(0, \nu_p(a_i) - \nu_p(a_{i+1})$ операции в a_i. Така може да сумираме за всяко p и всяко i тази бройка. С цел да бъде бързо, за съседни числа може да разгледаме само простите делители на a_i (бързо намерени с Решето на Ератостен), вместо всяко просто число изобщо – отново използващо факта, че общия брой деления е малък.

Постигната сложност: $O(n \log_2(max\{a_i\}))$

Автор: Борис Михов