	На пълното решение	На частичните решения
Тагове	Обхождания на графи Disjoint set union Решето на ератостен	Броене `std::bitset`

Анализ

А GCD_4 кога?

Подзадача №1

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

Подзадача №2

Може лесно да се забележи, че може да се приложи операцията само и единствено на числа, които първоначално са били равни. Така отговорът ще е равен на броя различни числа в редицата. С броячен масив намираме броят им в редицата.

Постигната сложност: $O(N)$

Имплементация: `gcd3_10p.cpp`

Размишления върху операцията

Първият въпрос, който състезател трябва да си зададе, когато решава задачата е: "Кога две числа не са взаимно прости?". Техният НОД трябва да е $\neq 1$, следователно трябва да споделят делител > 1. Щом споделят делител > 1, то те ще споделят и прост делител, защото, ако $x \mid a, b$, то ще $\exists p$, такова, че p е просто и $p \mid x$. Щом $p \mid x$ и $x \mid a, b$, то $p \mid a, b$. От това следва, че операцията може да се приложи единствено върху двойки числа, които споделят прост делител. Така, ако представим всяко число a_i като множество t_i от прости делители, то $S = \{t_1, t_2, t_3, \ldots, t_N\}$ и операцията ни изглежда от следния вид:

- Избери двойка множества t_i и t_j от мултимножеството S, така че $t_i \cap t_j \neq \emptyset$.
- Премахни t_i и t_j в мултимножеството и добави $t_i \cup t_j$.

Може да забележим също, че винаги когато можем да приложим операцията, то ни е в интерес да го направим. Ако допуснем противното ще следва, че Б.О.О. при избора на три множества t_i, t_j, t_k, то ще може да се приложи операцията върху двойките (t_i, t_j) и (t_i, t_k), но след прилагането на операцията върху t_i и t_j, няма да може да се приложи върху резултантното множество от операцията и t_k. Всъщност, ние ще целим да намерим тройка множества (t_i, t_j, t_k), за които $t_i \cap t_k \neq \emptyset$ и $(t_i \cup t_j) \cap t_k = \emptyset$, което е невъзможно.

Така един поглед върху задачата е следният – програмата ни трябва да търси двойки множества, и докато съществуват такива, да ги изтриваме и заменяме с обединението им. Така може да започнем с първите решения.

Подзадача №3

Има най-разнообразни подходи, по които може да решим подзадачата, но аз ще покажа по-интересен. Тъй като броят на простите числа до 10^4 е малък (около 1230), то може умно да приложим идеята от горния параграф. За всяко число отбелязваме в булев масив кои прости числа съдържа. Така две множеста ще имат сечение, когато имат единица на една и съща позиция в булевия масив. Нека намерим с кои множества първото има сечение. Ако съществува поне 1, то ще ”вмъкнем” първото в някое друго, тъй че може да отбележим, че броят на числата в редицата намалява с 1. Така, след като сме намерили множествата със сечение, ние може да отбележим изкувствено, че всички тези множества притежават делителите и на първото, като по този начин в последствие ще ги мърджнем. Продължаваме същия алгоритъм и с второто число в редицата, като вече гледаме множествата след него.

Постигната сложност: $O(N^2\,\pi(max(a_1, a_2, \ldots, a_N)))$

Имплементация: `gcd3_15p.cpp`

Подзадача №4

Всъщност, операциите за сечение и обединение на множества отговарят на съответно *побитово и* и *побитово или* в езика на програмирането. Чрез `std::bitset` вместо булев масив може да използваме побитовите операции, като за проверка дали две множества $bs[i]$ и $bs[j]$ имат сечение се използва `(bs[i] & bs[j]).count()`, а за намиране на сечението им – `bs[i] | bs[j]`. Така решението ни остава със същата сложност, но броя операции се дели на 64 от използването на битсет.

Постигната сложност: $O(N^2\,\pi(max(a_1, a_2, \ldots, a_N)))$

Имплементация: `gcd3_30p.cpp`

Cheat за подзадача №5

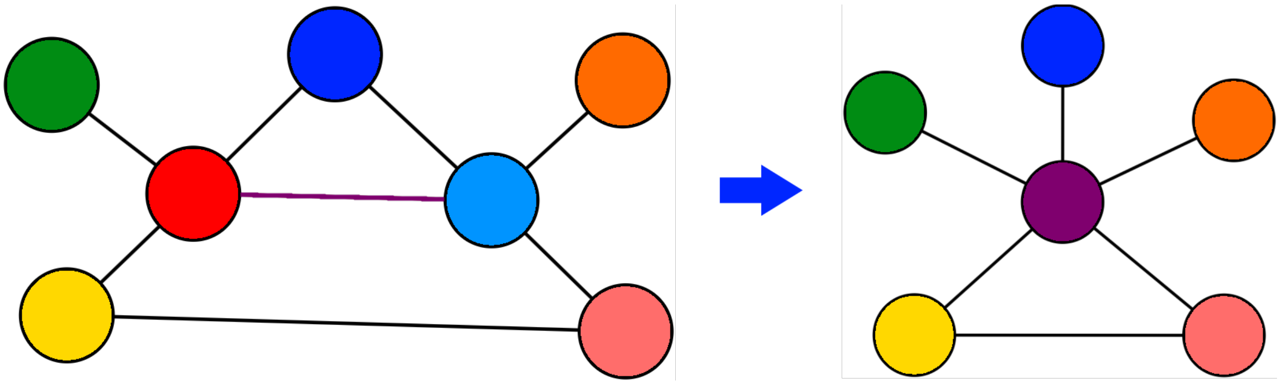
За пета подзадача може да забележим, че авторът е прецакан и не може да направи тест с много прости числа – има до $N + constant$ различни прости числа в тест. Така може да направим битсетовете си по-малки и с разделянето на 64 да мине за петата подзадача.

Постигната сложност: $O(N^3)$

Имплементация: `gcd3cheat_45p.cpp`

Графова идея

За по-умелите състезатели би дошло естествено структура от `позволеност/непозволеност` между двойки елементи да се разгледа като граф. И в тази задача, подобно разглеждане се оказва много удобно. Нека построим граф, в който съществува ребро между всяка двойка множества, които имат сечение. Как ще изглежда операцията в този граф?



При прилагане на операцията върху два върха в графа се премахват двата върха и се заменят с нов, на когото съседите му са обединиението на съседите на премахнатите върхове. Така се пита колко е минималният брой върхове, до които може да се стигне. И той с, това свеждане, става равен на броя компоненти в графа. Така всяко решение до момента имплицитно обхождаше графа.

Подзадача №5

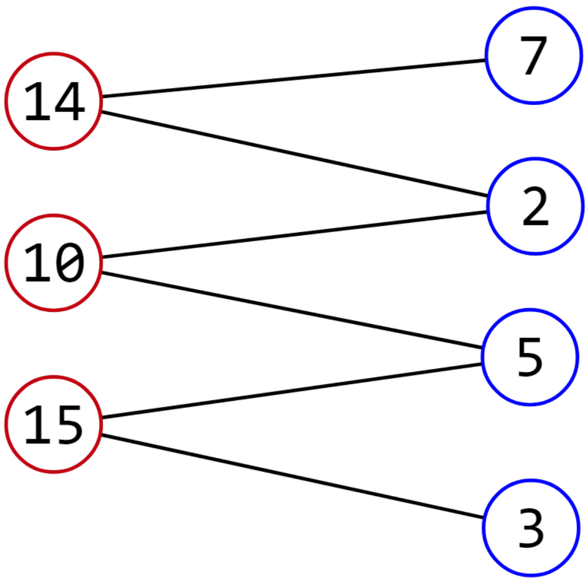
Построяваме споменатия граф, като използваме алгоритъм на Евклид за проверка дали две числа са взаимно прости. Използваме DFS или DSU за пресмятането на броя компоненти.

Постигната сложност: $O(N^2 log_2(max(a_1, a_2, ..., a_N)))$

Имплементация: `gcd3_45p.cpp`

Конструкция

За да се изкарат повече точки е нужна по-добра конструкция на графа. Към момента графа може да има $O(N^2)$ ребра, което е доста излишно – нас ни интересува единствено и само непряката свързаност. Тъй като две числа не са взаимно прости тогава и само тогава, когато споделят общ делител, ние може да конструираме следния еквивалентен на текущия граф като свързаност – създаваме фиктивни върхове за всяко просто число от 1 до 10^7 и свързваме всеки връх от редицата със съответните му прости делители. Така, ако в първоначалния граф е имало ребро между два върха, то сега те споделят съсед. Графът за $\{10, 14, 15\}$ е показан отдолу, където червените върхове са числата от редицата, а сините – простите числа. В графа ще има $O(N\ log_2(max(a_1, a_2, ..., a_N)))$ ребра, като задачата ни продължава да е да се изчисли броя компоненти. Единствено остава да се намерят простите делители на числата.



Подзадача №6

За всяко число прилагаме стандартното обхождане до корен за разлагане. Намираме броя компоненти чрез DFS или DSU.

Постигната сложност: $O(N\sqrt{max(a_1, a_2, \ldots, a_N)})$

Имплементация: `gcd3_75p.cpp`

Подзадача №7

Чрез Решето на Ератостен може да преизчислим най-малкия делител на всяко число и по този начин да разлагаме със сложност, която в най-лошият случай е log_2 от числото. Намираме броя компоненти чрез DFS или DSU.

Постигната сложност: $O(N\ log_2(max(a_1, a_2, \ldots, a_N)))$

Имплементация: `gcd3_100p.cpp`

Автор: Борис Михов