Анализ на задача ntmst

Задачата изиска да се намери максимално покриващо дърво в пълен граф, генериран чрез специални правила - тежестта на реброто между u и v е $\gcd(a_u, a_v)$.

Първа подзадача

Тук се очаква единствено познание на някой алгоритъм за намиране на минималното покриващо дърво (Прим-Ярник или Крускал). Технически алгоритмите са за *минимално* покриващо дърво, но като заместим всяко ребро с тежест x с такова с тежест $-x$ лесно може да се убедим, че алгоритъмът ще ни дава и *максималното* покриващо дърво.

Втора подзадача

Допълнителното условие значително улеснява задачата: за две прости числа x и y е вярно, че:

- $gcd(x, y) = x = y$, ако $x = y$;
- $gcd(x, y) = 1$, ако $x \neq y$

След като търсим някакъв максимум, видимо е оптимално да свързваме две равни числа, докато това е възможно. Когато всички такива компоненти са свързани, ще добавяме ребра с тежест 1, докато свържем целия граф.

Подзадачата беше дадена, за да се позволи допълнително изявяване на уменията за хората, които не успяват да решат цялата задача.

Трета подзадача

Стесненото ограничение върху стойностите в редицата $\{a\}_{i=1}^{n}$, налага въпросът за множество дублиращи се стойности - по принципа на Дирихле става очевидно, че за $n = 10^6$ и $a_i \leq 10^3$ ще има много повтарящи се стойности.

Това обаче е хубаво нещо (относно максимума) – ако имам x два пъти в редицата, то оптимално решение е да свържа двете срещания, защото $\gcd(x, x) = x$ и няма как $\gcd(x, y) > x$ за каквото и да е y. След като "елиминараме" дублиращите стойности, ще останем с най-много 10^3 различни числа в редицата - вече можем да пуснем решението на първа подзадача и да получим търсения отговор.

Четвърта подзадача

Тук вече е полезно да си дадем някаква по-дълбока идея на задачата, която решаваме - независимо от страничните детайли, **`ntmst`** е задача за минимално покриващо дърво. Такава задача трябва да се решава с алгоритъм за минимално покриващо дърво (Прим-Ярник или Крускал). Директно ще обясним решението чрез алгоритъма на Крускал, макар че по време на състезанието ще е добре участник да отдели малко време да опита да модифицира алгоритъма на Прим-Ярник.

Алгоритъмът на Крускал представлява многократно повтаряне на следната "лакома стъпка" – намираме най-лекото (при нас, тежкото) ребро, което ще свърже две несвързани компоненти в графа и го добавяме. В текущата ситуация имаме твърде много ребра, за да ги сортираме (стандартния метод за намиране на нужното ребро), но тежестите на ребрата се определят чрез ясно определено правило в задачата – ще се опитаме да се възползваме от него, за да улесним нужната заявка.

Тежестите на ребрата са ясно ограничение от $a_i \leq 10^6$ – няма как да взимаме НОД на две такива числа и да очакваме да получим резултат по-голям от 10^6. Ще започнем да търсим ребра с тежест d, като d го итерираме от 10^6 към 1. За да има реброто тежест d, трябва a_u и a_v да се делят на d - следователно можем да обиколим всички върхове u, за които a_u е кратно на d и

да се опитаме да ги свържем на верижка в една `union-find` структура. Забележете, че няма нужда да се тревожим за по-оптимален начин да свързваме върховете – вече сме фиксирали тежестта на ребрата, които ще добавяме, ако два върха можеха да се свържат с ребро с тежест по-голяма от d, алгоритъмът вече щеше да ги е свързал.

Сложността на решението е $O(n\sqrt{n}\alpha(n))$ – за всяко a_i, за всеки негов делител сме се опитали да създадам някакво ребро с него в *union-find* структурата. $\alpha(n)$ тук е обърнатата Акерманова функция.

Пета подзадача

Четвъртата подзадача всъщност ни свърши повечето от мисленето по задачата. Решението там с право прилича доста на решето на Ератостен, което обикновено се държи със сложност $O(n \log n)$, така че си струва човек да се опита да го дооптимизира. На по-човешки език, проблемът с горното решение е, че една и съща стойност може да се добави множество пъти и за всяко нейно добавяне трябва да обхождаме нейните делители. Проблемът с множеството еднакви стойности обаче решихме в трета подзадача – доказуемо е, че можем да добавим ребрата помежду им в оптималното решение. Премахвайки дубликатите и пускайки абсолютно същото решение ще получим алгоритъм със сложност $O(n \log n\alpha(n))$.

Полезни изводи от тази задача, които може човек да си направи са:

- Стандартна задача, независимо с колко допълнителни усложнения и улеснения, все още е стандартна задача и учебникарските подходи и алгоритми по нея най-вероятно ще дават меродавно решение. Тук независимо колко променена е ситуацията с ребрата, алгоритъмът на Крускал все даваше решение.
- Анализът на сложността на решето на Ератостен силно се възползва от нуждата да обработи делителите на всяко число *точно* веднъж. Много приятно тук успяхме да докажем ненужността на дублиращите се елементи. Други задачи не са такива късметлии – divide C1 ЕТИ 2022 примерно се нуждае от неестественото ограничение, че всяка стойност се среща по веднъж в редицата, за да има решение $O(n \log n)$.

Автор: Иван Лупов