Анализ на задача gcd_subsequence

Първа подзадача

Ниските ограничения по n позволяват решения базирани на метода "пълно изчерпване". Авторовата имплементация върви със сложност $O(2^n n \log(A))$.

Втора подзадача

Тази подзадача цели да възнагради състезателите с познания по метода "динамично оптимиране". Често срещан стейт е dp_i, който решава по-малка версия на поставената задачата за числата от префикса $[1, i]$ - тук той се оказва напълно достатъчен. Естественият транзишън между стейтовете е:

$$dp_i = \max_{j=1}^{i-1}(dp_j + \gcd(a_i, a_j))$$

Директната му имплементация със сложност $O(n^2)$ решава подзадачата.

Трета подзадача

С тази и следващата подзадача предоставям на шанса да се изкарват точки без да се решава цялата задача и състезателите постепенно да изграждат интуиция, водеща ги към пълното решение.

Тривиално ясно е, че за a_i просто число имаме две опции - или да го свържем с предходно равно число и да получим $+a_i$ към сумата ни, или с някое друго и да получим $+1$. Оптимално е и в двата случая да се връзваме към най-близкото предишно число, което отговаря на съответното условие. При $+1$ това винаги ще е числото на позиция $i-1$ (реално може да се окаже, че $a_{i-1} = a_i$, но това не наранява сумата ни), а за $+a_i$ ще поддържаме масив с последните срещания на всяко число.

Финалната сложност възлиза на $O(n)$.

Четвъртата подзадача

Тук отново ни се дава по-стриктно условие върху въведените стойности. Решението ще е подобно на това за втората подзадача, но вместо да обикаляме всички предходни позиции, ще обикаляме само най-близките предходни позиции за всяко число - ако имаме три двойки на позиции $2, 3$ и 5, няма как отговорът за позиции 2 и 3 да е по-добър от този за позиция 5.

Пълно решение

Оказва се, че функцията gcd предразполага по-стриктни ограничения, от които можем да се възползваме.

Независимо какво число изберем за x, то стойността на $\gcd(x, a_i)$ ще бъде някакъв делител на a_i. Ще фиксираме кой точно делител d искаме да бъде стойността на НОД-а. Сега обаче трябва да се погрижим d също да бъде делител на $x = a_j$. Ако тепърва почнем да търсим подходящите позиции в редицата ще вдигнем твърде много сложността (забележете, че аргументи от типа на "ще взема най-близката позиция"вече не работят - контрапримерът е упражнение за читателя).

За да се справим с горния проблем, ще погледнем на всяко число от подредицата по два начина: Ако взема число a_i в подредицата:

- то веднъж в решението a_i ще трябва да си търси най-оптималния предшественик;
- също така позициите след i-тата ще трябва някак лесно да достъпват информацията за тази позиция - конкретно ще ни интересува dp_i.

Това, което изяснихме е как ще се оправяме с първата роля на числото - ще фиксираме няка-

към делител на a_i и ще проверим сред предходните позиции коя е оптималната. Сега трябва да измислим удобно решение на втория проблем, тоест, такова което предполага нужната информация на по ясно достъпен начин.

Авторовото решение е за делител d в допълнителен масив да се поддържа максималното dp_i за някое i, такова че a_i се дели на d. Това представлява $O(\sqrt{A})$ промени в паметта.

Финалната сложност на решението за задачата е $O(n\sqrt{A})$.

Автор: Иван Лупов