Задача A. Тура

Можна побачити, що відповідь — це $n + m - 2$.

Задача B. Кімната

Відповідь на задачу — $\lfloor n/k \rfloor \times \lfloor m/k \rfloor$.

Задача C. Плавний контест

Треба перевірити, що $|a - b| \leqslant 1$, $|b - c| \leqslant 1$, $|c - d| \leqslant 1$ та $|d - e| \leqslant 1$.

Задача D. Пограбування масиву

Спочатку перевіримо другу умову, тому що якщо був відсортований до видалення, то й після нього він залишиться відсортованим. Нехай sum — сума елементів масиву B. Розглянемо два випадки:

- $sum \bmod 2 = 0$ — щоб $sum \bmod 2 = 0$ залишалося й після додавання елементу, треба додавати 0. Нехай cnt_0 — кількість 0 в масиві B, тоді ми можемо обрати $cnt_0 + 1$ позицію, щоб поставити новий елемент. $ans = cnt_0 + 2$.
- $sum \bmod 2 = 1$ — щоб стало $sum \bmod 2 = 0$ після додавання елементу, треба додавати 1. Нехай cnt_1 — кількість 1 в масиві B, тоді ми можемо обрати $cnt_1 + 1$ позицію, щоб поставити новий елемент. $ans = cnt_1 + 1$

Задача E. Цукеркоман

Треба завжди вибирати найменший доступний елемент. Якщо є деяка оптимальна множина A, яка не складається з найменших можливих елементів, то завжди можна взяти найбільший елемент з A і замінити його на найменший доступний. Порахуймо масив cnt_i — скільки разів число i входить в інтервали. Використаємо метод скануючої прямої. Створимо масив b, для кожного інтервалу додамо до b_{l_i} 1, та віднімемо 1 від b_{r_i+1}. Можна помітити, що масив cnt це масив префікс сум на масиві b, тобто, $cnt_i = \sum_{j=1}^{i} b_j$. Тоді будемо йти від 1 до 10^5. Нехай $take = \min(\lfloor w/i \rfloor, i, cnt_i)$, тоді додаємо до відповіді $take$, та робимо $w = w - take \cdot i$.

Задача F. Гра-гра-граф

У грі перший гравець виграє лише за умови, що початковий граф — повний, або йому не вистачає одного ребра. Якщо в графі не буде більше одного ребра, то на кожний крок додавання другий гравець може відповідати видаленням цього ребра.

Граф на n вершинах називається повним, якщо він має $\frac{n\cdot(n-1)}{2}$ ребер.

Граф на n вершинах назвемо виграшним, якщо $\frac{n\cdot(n-1)}{2} \leqslant m+1$, де m — кількість ребер в графі.

Граф назвемо програшним, якщо він не є виграшним.

Рішення за $\mathcal{O}(n^3+n^2\cdot m)$ — перебрати всі відрізки та порахувати кількість ребер, які сполучають вершини з даного відрізку. Набиратиме близько 35 балів.

Рішення за $\mathcal{O}(n^2+n\cdot m)$ — будемо перебирати ліву границю та збільшуючи праву границю одразу рахувати кількість ребер в даному графі.

Рішення за $\mathcal{O}(n + m\sqrt{m})$ — можна помітити, що якщо граф не був виграшним для відрізка $[l;r]$, то він не стане виграшним для відрізка $[l;r+1]$. Якщо граф стає програшним, то можна одразу виходити з циклу. Таке маленьке спостереження пришвидшує рішення до вказаної вище асимптотики.

Рішення за $\mathcal{O}(n+m)$ — використаємо техніку двох вказівників. Рахуватимемо кількість виграшних графів з лівою границею в l. Щоб перевіряти, чи є граф виграшним, будемо підтримувати кількість ребер у даному підграфі. Орієнтуємо ребра в графі від вершини з більшим індексом до вершини з меншим індексом і будемо підтримувати cnt_v — кількість ребер, які йдуть від вершини v і є в даному графі. Помітимо, що cnt_v треба підтримувати тільки якщо вершина v є в графі. Поки граф буде виграшним, будемо зсувати праву границю на одну позицію вправо і перераховувати cnt_v. Кількість виграшних графів з лівою границею в l дорівнює $r-l$, оскільки ми маємо півінтервал $[l;r)$. Коли зсуваємо ліву границю, віднімаємо від кількості ребер cnt_l.

Посилання на тему "два вказівники": `http://bit.ly/3Xo5SVo`.

Доведення асимптотики третього рішення залишається учасникам.

Задача G. AND-шлях

Напишемо функцію $check(x)$, яка буде перевіряти, чи може x бути відповіддю. Зробимо $dp_{i,j} \in [0, 1]$ в залежності від того, чи можемо/не можемо дістатися до цієї клітинки. Тобто, $dp_{i,j} = 1$ якщо можна дістатися до клітинки $(i;j)$, і $dp_{i,j} = 0$ інакше. Також ми можемо використовувати лише ті клітинки, для яких x є підмаскою їх числового значення. Число A є підмаскою числа B якщо (A `AND` B) $= A$, де `AND` — побітове І. Тоді маємо наступну формулу:

$$dp_{i,j} = dp_{i,j-1} \ \ OR \ \ dp_{i-1,j}$$

Якщо $dp_{n,m} = 1$, то ми можемо отримати відповідь x, інакше — ні.

Будемо перебирати біти у відповіді від старших до молодших. Тобто, спочатку буде йти біт, який відповідає за 32 у числі, потім біт, який відповідає за 16 і так далі. Спробуємо додати біт у відповідь, якщо вийшло, то додаємо, інакше нічого не робимо — $if(check(ans + 2^i))\ ans = ans + 2^i$.

Асимптотика рішення — $\mathcal{O}(n \cdot m \cdot \log A)$, де A — максимальне значення чисел у вхідних даних.

Задача H. Знову запити?...

Ми можемо отримати gcd рівне d, якщо існують хоча б два додатних числа у масиві, які діляться на d. Розіб'ємо числа на дільники та будемо зберігати їх кількість.

Помітимо, що кожне число змінюється максимум $\log A$ разів, де A — максимальне значення числа. Це нескладно довести, тому що якщо біт був ввімкнутий (1), то він або може залишитися увімкнутим, або стати вимкнутим (0). Якщо біт був вимкнутий, він не може стати ввімкнутим. Кожен біт може змінити свій стан рівно один раз, а отже кількість змін числа буде не більше кількості увімкнутих бітів в числі — $\log A$.

Навчімося змінювати числа рівно $\log A$ разів. Нам треба придумати якусь умову, щоб якщо не було такого числа, яке треба оновлювати, то ми не оновлюємо жодне. Якщо існує якийсь біт, якого нема в числі x в запиті оновлення, а у масиві він є, то треба його вимкнути та оновити число. Можна легко перевірити, що нам нічого оновлювати не треба. Порахуємо val_{OR} = побітове АБО (`OR`) чисел. Якщо (val_{OR} `AND` x) = val_{OR}, то не існує такого біта, який треба оновлювати.

Напишемо дерево відрізків, де для кожного відрізку будемо підтримувати значення побітового АБО (`OR`) чисел на ньому. Обробляючи кожен запит будемо спускатися по дереву відрізків. Якщо на цьому відрізку нам нема що оновлювати — не йдемо далі. Якщо ми прийшли у лист дерева, то оновимо число — видалимо усі дільники числа a_i, та додамо дільники числа a_i `AND` x, де x — число з запиту оновлення.

Щоб підтримувати відповідь для масиву, можна скористуватися структурою `set`. Якщо кількість якогось дільника стала дорівнювати двом, то додамо дільник в `set`, а якщо вона стала дорівнювати одному і до цього дорівнювала двом, то видалимо це число з `set`. Відповідь на запит — найбільше число в сеті, чи 0, якщо `set` не містить жодного числа.

Асимптотика рішення — $\mathcal{O}(n \cdot \log n \cdot \log A + n \cdot d)$, де A — максимальний елемент в масиві, а d — максимальна кількість дільників у числа до A.

Лекція на тему Дерево відрізків від наших друзів з `Algotester`: `http://bit.ly/3HgOEDF`.

Автор усіх задач: Андрій Столітній.