Задача A. Пиріжки

Кожен з m друзів Козака Вуса хоче рівно три пиріжки, тобто разом вони хочуть $3m$ пиріжків.

Оскільки Козак Вус віддає $3m$ зі своїх n пиріжків, у нього залишається $n - 3m$ з них. Отже, достатньо вивести $n - 3m$.

Задача B. Анфіса і квіти

Ми можемо зауважити, що після першого ходу Анфіса збирає принаймні $n+m-1$ квітів, оскільки найкоротший шлях від $(1, 1)$ до (n, m) проходить принаймні через таку кількість клітин. Також, після кожного наступного ходу, вона збирає принаймні ще одну квітку.

Оскільки загальна кількість квітів дорівнює $n \cdot m$, це вже дає верхню межу на відповідь як $nm - (n + m - 1) + 1 = (n - 1)(m - 1) + 1$.

Водночас, ми можемо показати, що Анфіса завжди може гарантувати отримання принаймні такої кількості шматочків сиру: на своєму першому ході вона переміщується з $(1, 1)$ до $(n, 1)$ (тільки вертикально) і з $(n, 1)$ до (n, m) (тільки горизонтально), збираючи $n + m - 1$ квітів.

На кожному з її наступних ходів вона намагатиметься зібрати лише одну нову квітку — нехай (x, y) буде найнижчою з найлівіших квітів. Анфіса тоді рухається прямою лінією з $(1, 1)$ до $(x, 1)$, з $(x, 1)$ до (x, y), з (x, y) до (n, y) і з (n, y) до (n, m). Оскільки (x, y) була найнижчою з найлівіших квітів, на її шляху немає інших квітів, і, отже, вона збирає точно одну квітку під час цього ходу. Тому, оскільки після першого ходу залишилося $nm-(n+m-1) = (n-1)(m-1)$ квітів, Анфіса має стратегію виконати принаймні $(n - 1)(m - 1)$ додаткових ходів, що в сумі становить $(n - 1)(m - 1) + 1$ ходів.

Отже, відповідь дорівнює $(n - 1)(m - 1) + 1$.

Задача C. НМТ

Автор, розробник, автор розбору: Андрій Столітній

Потрібно перевірити, чи $p_1w_1 + p_2w_2 + p_3w_3 + p_4w_4 \geqslant S$. Якщо так, вивести “YES”, інакше “NO”.

Задача D. Змагання

Автор, розробник, автор розбору: Андрій Столітній

Спочатку треба посортувати події неспадаючи за часом. Далі можемо підтримувати момент, коли команда вийшла, або “-1”, якщо команда зараз присутня на виступі. Коли приходить нова подія $(t; a)$ для команди a, треба перевірити, чи $t-last_a > X$, якщо $last_a \neq -1$, та присвоїти $last_a := t$ інакше. Якщо

для команди було виконано $t - last_a > X$, то вона стає дискваліфікованою. Множину дискваліфікованих команд можна підтримувати за допомогою масиву з станами 0/1.

Задача E. Видалення тестів

Автор, розробник, автор розбору: Тимур Деґтярь

Для тестів, що задовольняють $n, m \leqslant 50$, можна використати повний перебір: ми перебираємо всі можливі значення d (це $1, 2, \ldots, m-1$, зауважимо, що при $d \geqslant m$ Містер IP ніколи не видалить жоден тест), і для кожної пари (i, j) перебираємо всі $1 \leqslant k \leqslant m$ та явно рахуємо, скільки з них задовольняють $a_{i,k} > a_{i,j}$, і відмічаємо, чи є тест j "недійсним"для рішення i при даному значенні d.

Після цього залишається лише підрахувати, скільки тестів буде видалено, щоб перевірити, чи підходить це значення d. Обравши максимальне d, яке видаляє достатньо тестів, ми знаходимо відповідь. Якщо жодне значення d не підходить, виводимо -1. Часова складність такого рішення — $O(nm^3)$.

Для тестів, що задовольняють $n, m \leqslant 400$, можна покращити описане вище рішення. Якщо для кожної пари (i, j) попередньо порахувати, скільки існує таких k, що $a_{i,k} > a_{i,j}$, то для фіксованих d та пари (i, j) можна визначити за $O(1)$, чи є тест j "недійсним"для рішення i. Попереднє обчислення можна виконати наївно — для кожної пари (i, j) перебираємо всі $1 \leqslant k \leqslant m$. Це робиться за $O(nm^2)$. Перевірка кожного значення d займає $O(nm)$ часу, і оскільки існує $m - 1$ значень d, які потрібно перевірити, загальна складність становить $O(nm^2)$, а отже, рішення у цілому працює за $O(nm^2)$.

Для повного розв'язку потрібно оптимізувати дві речі — обчислення для кожної пари (i, j) кількості таких k, що $a_{i,k} > a_{i,j}$, та пошук відповідного значення d. Для першої частини можна для кожного i відсортувати масив пар $a[i][j], j$, після чого кількість таких k буде рівною $m-$(позиція j у відсортованому масиві). Для другої частини можна зробити таке спостереження: якщо для деякого значення d та рішення i тест j є "недійсним то він буде "недійсним"і для $d-1$. Звідси випливає: якщо для деякого d тест j був видалений, то для $d-1$ його також буде видалено. Це дозволяє виконати бінарний пошук за значенням d: встановлюємо межі пошуку $[0, , m-1]$, і якщо результат — 0, то відповідь -1, інакше ми знайдемо її бінарним пошуком.

Задача F. Банк

Автор, розробник, автор розбору: Тимур Деґтярь

Ключове спостереження (1) для цієї задачі полягає в тому, що кожна степінь числа 2 може бути складена як сума менших степенів числа 2.

Оскільки порядок елементів масиву a не має значення, ми можемо відсортувати його у спадному порядку і вважати, що a відсортований.

Використовуючи спостереження (1), отримаємо спостереження (2). Нехай існує деякий масив b довжини m з властивістю $b_i \geqslant b_{i+1}$ для всіх $1 \leqslant i \leqslant m-1$, і $S = S = 2^{b_1} + 2^{b_2} + 2^{b_3} + \cdots + 2^{b_m}$. Тоді кожне число з $2^{b_1}, 2^{b_1+1}, \ldots, 2^{\lfloor \log_2(S) \rfloor}$ можна виразити як префіксну суму масиву b.

Нехай $S = 2^{a_1} + 2^{a_2} + 2^{a_3} + \cdots + 2^{a_n}$ — загальна сума всіх витрат.

Позначимо через $Ans(a)$ найменшу можливу найбільшу суму 2^{a_j} в одній категорії (тобто оптимальне значення, яке ми хочемо мінімізувати).

Якщо $S < k \cdot 2^{a_1}$, тоді $Ans(a) = 2^{a_1}$.

Оскільки принаймні одна категорія міститиме елемент 2^{a_1}, тому відповідь не може бути меншою за 2^{a_1}. З іншого боку, для кожної з інших категорій ми можемо поступово додавати елементи з масиву a (у порядку спадання) доти, доки сума в цій категорії не стане рівною 2^{a_1} або ми не вичерпаємо всі елементи з a. За спостереженням (2), це завжди можливо зробити, не перевищуючи 2^{a_1}.

Якщо $S \geqslant k \cdot 2^{a_1}$:

Розглянемо деяке оптимальне розбиття елементів масиву a на k категорій. Без втрати загальності будемо вважати, що перша категорія має найбільшу суму. Із кожної категорії візьмемо деяку підмножину елементів, яка в сумі дорівнює 2^{a_1} (або всі елементи з цієї категорії, якщо їхня сума менша за 2^{a_1}), і видалимо ці елементи з масиву a. Позначимо отриманий після цього масив як a'.

Очевидно, що перша категорія і надалі має найбільшу суму. Також очевидно, що $Ans(a) = 2^{a_1} + Ans(a')$, оскільки інакше можна було б перерозподілити елементи між категоріями так, щоб ця рівність стала істинною в обидві сторони, що суперечить оптимальності.

Крім того, якщо для деякої категорії сума її елементів менша за 2^{a_1}, тоді a' мусить бути порожнім: в іншому випадку ми могли б перекинути частину елементів з a' у цю категорію і покращити (зменшити) максимальну суму, знову ж суперечачи оптимальності розбиття.

Позначимо через $M = Ans(a)$, а через $S = \sum_{i=1}^{n} 2^{a_i}$ — початкову загальну суму. Повторно застосовуючи перехід $a := a'$, ми зрештою потрапимо в ситуацію, коли вже для деякого масиву a' буде виконано $2^{a'_1} \cdot k > \sum_i 2^{a'_i}$, тобто ми опинимось в випадку 1 для масиву a' і тоді $Ans(a') = 2^{a'_1}$. Оскільки на кожному кроці ми "знімаємо"з кожної категорії по сумі 2^{a_1}, загальна величина M дорівнюватиме $M = \frac{(S - \sum 2^{a'_i})}{k} + 2^{a'_1}$, отже, оптимально мінімізувати найбільший елемент a', для чого достатньо використовувати префікс a, щоб сформувати k категорій зі сумами 2^{a_1}.

Таким чином, ми отримуємо жадібне рішення, яке виконується наступним чином:

Відсортуйте a у спадному порядку, і поки він не порожній, якщо сума 2^{a_i} менша за $k \cdot 2^{a_1}$, призначте категорії таким чином, щоб жодна сума не перевищувала 2^{a_i} і завершіть. В іншому випадку, розділіть деякий префікс a на k категорій зі сумами рівно 2^{a_i}, видаліть цей префікс і продовжуйте.

Остаточна часова складність цього рішення, за правильної реалізації, становить $\mathcal{O}(n \log n)$.

Задача G. Матриця

Автор, розробник, автор розбору: Андрій Столітній

Для фіксованого стану матриці задачу можна розв'язувати за допомогою динамічного програмування. Нехай $dp_{i,j}$ — сторона найбільшої квадратної матриці, яка складається тільки з одиниць та має правий нижній кут в $(i; j)$. Тоді можна зробити перерахунок значень наступним чином:

$$dp_{i,j} = \begin{cases} 0, & \text{якщо } a_{i,j} = 0, \\ 1 + \min\ (dp_{i-1,j-1},\ dp_{i-1,j},\ dp_{i,j-1}), & \text{якщо } a_{i,j} = 1. \end{cases}$$

А відповідь тоді буде максимумом по всіх значеннях $(i; j)$, $ans = \max\limits_{1 \leqslant i,j \leqslant n} dp_{i,j}$.

Можна помітити, що $0 \leqslant dp_{i,j} \leqslant n$ для всіх позицій $(i; j)$. Друга важлива річ, що запити тільки можуть поміняти 0 на 1, тобто відповідь ніколи не погіршується, тобто кожне зі значень $dp_{i,j}$ ніколи

не зменшується. Отже, можемо оновлювати значення dp частково. Якщо ми оновили значення $dp_{i,j}$, то воно може вплинути тільки на значення $dp_{i,j+1}, dp_{i+1,j}, dp_{i+1,j+1}$. Можемо перераховувати значення $(i; j)$, і якщо воно змінилось, перерахуймо значення, які залежать від нього. Якщо вони змінились, то перерахуємо значення, які залежать від них і так далі. Це можна реалізувати або чергою, або за допомогою рекурсії. Так як кожне значення dp може змінюватись не більше ніж n разів, а таких значень маємо n^2, то загальний час виконання програми буде $\mathcal{O}(n^3)$.

Задача H. Мін-Макс-Множення!

Автор: Тимур Деґтярь
Розробники: Андрій Столітній, Тимур Деґтярь
Автор розбору: Андрій Столітній

Ми хочемо порахувати

$$S = \sum_{l=1}^{n} \sum_{r=l}^{n} \mathrm{mn}(l, r) \cdot \mathrm{mx}(l, r)$$

за $\mathcal{O}(n \log n)$.

Внесок підмасивів довжини 1 легко рахується окремо як

$$\sum_{i=1}^{n} a_i^2.$$

Далі розглядаємо тільки підмасиви довжини $\geqslant 2$.

Напишемо функцію solve(l, r), яка рахує суму для всіх підмасивів, що **повністю** лежать у $[l, r)$. Нехай $m = \lfloor (l + r)/2 \rfloor$. Тоді всі підмасиви діляться на:

1. повністю в $[l, m)$;
2. повністю в $[m, r)$;
3. такі, що перетинають середину: $i \leqslant m - 1$, $j \geqslant m$.

Перші два типи рахуємо рекурсивними викликами solve(l, m) та solve(m, r). Залишається навчитися ефективно рахувати внесок підмасивів, що перетинають середину.

Для відрізка $[m, r)$ йдемо $j = m, \ldots, r - 1$, підтримуємо

$$mn_R(j) = \min(a_m, \ldots, a_j), \quad mx_R(j) = \max(a_m, \ldots, a_j),$$

та префікси

$$\mathrm{prmin}[k] = \sum_{t=m}^{k-1} mn_R(t), \quad \mathrm{prmax}[k] = \sum_{t=m}^{k-1} mx_R(t), \quad \mathrm{pr}[k] = \sum_{t=m}^{k-1} mn_R(t) \cdot mx_R(t),$$

де всі суми беруться за модулем.

Тоді, наприклад, $\mathrm{prmin}[q] - \mathrm{prmin}[p]$ дорівнює $\sum_{j=p}^{q-1} mn_R(j)$.

Перебираємо лівий кінець i від $m - 1$ до l (у зворотному напрямку), підтримуємо

$$mn_L = \min(a_i, \ldots, a_{m-1}), \quad mx_L = \max(a_i, \ldots, a_{m-1}).$$

Для цього i треба додати внесок усіх підмасивів $[i, j]$, де $j \in [m, r-1]$.

Використовуємо два вказівники по правій частині:

- p — перша позиція, де $a_p \leqslant mn_L$;
- q — перша позиція, де $a_q \geqslant mx_L$.

Далі маємо два випадки.

Випадок $p < q$

Тоді:

- для $j \in [m, p)$: $\text{mn} = mn_L$, $\text{mx} = mx_L$, внесок
$$(p - m) \cdot (mn_L \cdot mx_L);$$
- для $j \in [p, q)$: $\text{mn} = mn_R(j)$, $\text{mx} = mx_L$, внесок
$$mx_L \cdot \big(\text{prmin}[q] - \text{prmin}[p]\big);$$
- для $j \in [q, r)$: $\text{mn} = mn_R(j)$, $\text{mx} = mx_R(j)$, внесок
$$\text{pr}[r] - \text{pr}[q].$$

Випадок $p \geqslant q$

Аналогічно, але ролі мінімуму і максимуму частково міняються:

- для $j \in [m, q)$: і мінімум, і максимум задаються лівою частиною, внесок
$$(q - m) \cdot (mn_L \cdot mx_L);$$
- для $j \in [q, p)$: $\text{mn} = mn_L$, $\text{mx} = mx_R(j)$, внесок
$$mn_L \cdot \big(\text{prmax}[p] - \text{prmax}[q]\big);$$
- для $j \in [p, r)$: і мінімум, і максимум задаються правою частиною, внесок
$$\text{pr}[r] - \text{pr}[p].$$

Усі ці суми беруться з префіксів за $\mathcal{O}(1)$.

На кожному рівні рекурсії ми обробляємо елементи відрізка $[l, r)$ за $\mathcal{O}(r - l)$, глибина рекурсії — $\mathcal{O}(\log n)$, тому загальна асимптотика $\mathcal{O}(n \log n)$, а використана пам'ять — $\mathcal{O}(n)$.