Задача A. Унікальні літери

Автор задачі: Денисов Костянтин
Задачу підготував: Холод Андрій
Розбір написав: Холод Андрій

Блок 1

Якщо $|s| = |t| = 1$, то рядок s' складається з одного символу, і результат його унікалізації — це сам цей символ. Отже, відповідь YES тоді і тільки тоді, коли $s = t$.

Блок 2

Якщо s є перестановкою t, то $c_s[i] = c_t[i]$ для всіх i. Зокрема, у цьому випадку $|s| = |t|$, тому кожна літера зустрічається в s рівно стільки разів, скільки в t, тобто не більше одного разу. Але тоді жодних скасувань у процесі унікалізації не відбувається, і результат унікалізації — це сам рядок s' у тому порядку, в якому він записаний. Отже, достатньо взяти $s' = t' = t$ (або будь-яку перестановку t), і відповідь YES.

Блок 3

Якщо всі літери у s різні, то $c_s[i] \leqslant 1$ для всіх i. Оскільки жодна літера не повторюється, під час унікалізації ніколи не виникне ситуація, коли поточний символ вже є в a, тому скасувань не відбувається взагалі. Результат унікалізації — це просто сам рядок s' у тому порядку, в якому він записаний. Отже, задача зводиться до перевірки, чи s є перестановкою t: якщо так — відповідь YES і $s' = t'$ — будь-яка спільна перестановка, інакше — NO.

Блок 4

Є рівно одна літера A з $c_s[A] \geqslant 2$. Ключове спостереження: конструкція A $[\ldots]$ A у рядку s' «поглинає» все між двома входженнями A. Справді, коли ми обробляємо друге входження A, літера A вже є в рядку a, тому видаляється суфікс від першого входження A включно — тобто зникають A і всі літери після неї аж до поточної позиції.

Введемо поняття парності: для кожної літери i різниця $c_s[i] - c_t[i]$ має бути парною і невід'ємною. Невід'ємність очевидна — не можна отримати більше літер, ніж є у s. Парність пояснюється тим, що кожна «зайва» копія літери i (понад ту одну, що виживає в t) має бути скасована разом із якоюсь іншою копією тієї ж літери, утворюючи пару.

Якщо для всіх літер різниця парна — відповідь YES, виводимо відсортовані s і t. Якщо є літери з непарною різницею, їх треба «закрити» конструкцією A [непарні] A, де всередину поміщаємо рівно по одному екземпляру кожної непарної літери. Якщо після цього сам A лишається непарним (тобто $c_s[A] - c_t[A] - 2$ непарне) — відповідь NO, бо більше немає літер з достатньою кількістю копій для другого обгортання.

Блок 5

У цій підзадачі $s =$ `aaaaabbbbbccccc ddddd`, тобто всі літери згруповані блоками. Це дозволяє повністю зрозуміти загальний принцип. Умова парності $c_s[i] - c_t[i] \equiv 0 \pmod 2$ для всіх i є необхідною і достатньою для існування відповіді (за наявності хоча б одного обгортача). Якщо вона виконується — відсортований s є коректним s', оскільки всі зайві однакові літери стоять поруч і скасовуються парами. Інакше треба будувати обгортачі, як описано нижче.

Блок 6

Повністю вирішимо задачу для випадку, коли є рівно один обгортач. Спершу перевіримо необхідні умови: у рядку t кожна літера зустрічається не більше одного разу — інакше NO, бо результат унікалізації за визначенням не може містити повторів. Кожна літера з t має бути присутня у s — інакше NO. Для кожної літери i різниця $c_s[i] - c_t[i]$ має бути невід'ємною — інакше NO.

Збираємо список `odd` — літери з непарною різницею $c_s[i] - c_t[i]$, і список `have` — літери з $c_s[i] - c_t[i] \geqslant 2$. Якщо `odd` порожнє — виводимо відсортовані s і t. Якщо `odd` непорожнє, але `have` порожнє — відповідь NO. Якщо $|\texttt{have}| = 1$, беремо обгортач $A = \texttt{have}[0]$ і будуємо конструкцію A [усі непарні літери крім A] A. Якщо після цього сам A лишається непарним — відповідь NO, бо більше нема чим обгортати. Після побудови конструкції дописуємо в кінець усі літери, що залишились у s (вже відсортовано).

Блок 7

Розглянемо випадок, коли $|\texttt{have}| \geqslant 2$. Беремо два обгортачі $A = \texttt{have}[0]$ і $B = \texttt{have}[1]$ і будуємо: A [усі непарні літери крім A і B] A B [непарний залишок A якщо є] B. Перша пара закриває всі непарні літери крім B, а друга — можливий непарний залишок A. Двох обгортачів завжди достатньо, щоб закрити всі непарності, оскільки після першого обгортання непарними можуть лишитись лише A та B, а друге обгортання закриває їх обох. Після побудови конструкції дописуємо в кінець усі літери, що залишились у s (вже відсортовано).

Складність алгоритму — $\mathcal{O}(|s| \log |s|)$ через сортування рядка s, решта логіки працює за $\mathcal{O}(26)$.

Задача B. Перетин префіксних сум

Автор задачі: **Антон Тригуб**
Задачу підготував: **Андрій Куц**
Розбір написав: **Тарас Деркач**

Блок 1

У цьому блоці всі елементи масиву однакові. Нехай це значення дорівнює y.

Тоді порядок елементів не має значення. У будь-якій перестановці префіксні суми будуть такими:

$$y,\ 2y,\ 3y,\ \ldots,\ ny.$$

Отже, відповідь "NO"тоді й лише тоді, коли існує таке k $(1 \leqslant k \leqslant n)$, що $k \cdot y = x$.

Інакше можна вивести "YES"та початковий масив.

Блок 2

У цьому блоці $x = 0$.

Якщо сума всього масиву дорівнює 0, то відповідь "NO бо весь масив також є непорожнім префіксом.

Інакше позначимо суму всіх елементів як S.

Якщо $S > 0$, розташуємо елементи в порядку спадання. Тоді спочатку йдуть додатні елементи, а потім недодатні. Префіксна сума не може стати рівною 0, бо якби це сталося, то всі наступні елементи були б недодатними, а отже загальна сума не могла б бути додатною.

Якщо $S < 0$, аналогічно розташуємо елементи в порядку зростання.

Таким чином, у цьому блоці завжди можна побудувати відповідь, якщо $S \neq 0$.

Блок 3

Оскільки $n \leqslant 9$, можна перебрати всі перестановки масиву.

Для кожної перестановки рахуємо всі префіксні суми. Якщо жодна з них не дорівнює x, виводимо цю перестановку.

Складність такого розв'язку — $\mathcal{O}(n! \cdot n)$.

Блок 4

Для $n \leqslant 15$ можна використати динаміку по підмножинах.

Нехай $dp[mask]$ означає, що можна отримати підмножину елементів $mask$ як перші елементи перестановки так, щоб жодна префіксна сума не дорівнювала x.

Переходимо з $mask$ у $mask \cup \{i\}$, якщо сума елементів нової підмножини не дорівнює x.

Якщо можна отримати повну множину всіх елементів, то відповідь існує. Саму перестановку можна відновити за переходами динаміки.

Складність — $\mathcal{O}(n \cdot 2^n)$.

Блоки 5 та 8

У цих блоках усі елементи додатні.

Якщо сума всього масиву дорівнює x, то відповідь "NO бо повний масив є префіксом.

Якщо сума всього масиву менша за x, то будь-яка перестановка підходить, бо всі префіксні суми також будуть менші за x.

Залишається випадок, коли сума всього масиву більша за x.

Якщо серед додатних елементів є хоча б два різні значення, відсортуємо їх за зростанням. Якщо в такому порядку з'явився префікс із сумою x, то можна зіпсувати саме цей префікс: поміняти поточний елемент із найбільшим елементом або циклічно перенести найменший елемент у кінець.

Після такої зміни префікс, який раніше дорівнював x, стане або більшим за x, або меншим за x, але точно не рівним x. Оскільки всі елементи додатні, далі префіксні суми тільки зростають.

Якщо всі додатні елементи однакові та дорівнюють c, то всі префіксні суми мають вигляд $c, 2c, 3c, \ldots$. Тому якщо x ділиться на c, відповідь "NO інакше відповідь "YES".

Блок 6

У цьому блоці в масиві є не більше двох різних значень.

Цей випадок уже близький до повного розв'язку. Після сортування масиву достатньо розглянути, чи є серед додатних елементів два різні значення.

Якщо такі значення є, використовуємо ту саму ідею: спочатку ставимо додатні елементи, а один небезпечний префікс, якщо він з'явився, ламаємо перестановкою двох елементів.

Якщо всі додатні елементи однакові, задача зводиться до перевірки подільності на це значення. Саме цей випадок детальніше розглядається у повному розв'язку.

Блок 7

За умовою цього блока, якщо правильна перестановка існує, то її можна отримати, помінявши місцями перший елемент масиву з деяким іншим.

Тому потрібно перевірити всі такі перестановки.

Щоб зробити це швидко, порахуємо префіксні суми початкового масиву. Якщо ми міняємо місцями a_1 та a_i, то:

- перший префікс стає рівним a_i;

- префікси до позиції $i-1$ змінюються на $-a_1+a_i$;
- префікси після позиції i не змінюються.

Тому можна перебирати i та підтримувати множину потрібних префіксних сум. Якщо для деякого i після такого обміну жоден префікс не дорівнює x, виводимо цю перестановку.

Складність можна отримати $\mathcal{O}(n \log n)$.

Блок 9

У цьому блоці в масиві є рівно одне від'ємне число.

Якщо всі додатні числа не однакові, то працює ідея з додатними числами: ставимо додатні числа на початок, виправляємо можливий небезпечний префікс, а від'ємне число ставимо в кінець.

Якщо всі додатні числа однакові та дорівнюють c, то є два випадки.

Якщо x не ділиться на c, то префікси, які складаються лише з додатних чисел, не можуть дорівнювати x.

Якщо x ділиться на c, але від'ємне число не ділиться на c, то можна поставити це від'ємне число на початок. Після цього всі наступні префіксні суми матимуть інший залишок за модулем c, тому не зможуть дорівнювати x.

Якщо ж і x, і всі елементи масиву діляться на c, то після ділення на c усі додатні елементи стануть рівними 1. Щоб загальна сума стала більшою за x, у будь-який момент треба буде пройти через значення x, бо збільшення відбувається тільки на 1. Тому відповідь "NO".

Блок 10

Опишемо повний розв'язок.

Спочатку зробимо так, щоб $x \geqslant 0$. Якщо $x < 0$, помножимо всі елементи масиву та саме x на -1. У кінці, якщо ми змінили знак, потрібно вивести знайдену перестановку із поверненими назад знаками.

Нехай S — сума всіх елементів масиву.

Якщо $S = x$, то відповідь "NO бо весь масив є непорожнім префіксом.

Далі сортуємо масив за зростанням.

Якщо $x = 0$, використовуємо розв'язок з другого блока.

Якщо $S < x$, то можна вивести відсортований масив. Спочатку йдуть недодатні елементи, а потім додатні. Поки ми не дійшли до додатних чисел, префіксна сума не більша за 0. Після цього вона зростає, але фінальна сума все одно менша за x. Отже, префікс із сумою x не з'явиться.

Тепер залишився головний випадок:

$$0 < x < S.$$

Тоді в масиві обов'язково є хоча б одне додатне число.

Нехай mn — найменше додатне число, а mx — найбільше додатне число.

Випадок 1: $mn \neq mx$

Тобто серед додатних чисел є хоча б два різні значення.

Візьмемо всі додатні числа та відсортуємо їх за зростанням. Спочатку побудуємо перестановку тільки з них.

Якщо серед їхніх префіксних сум немає x, то все добре.

Якщо деякий префікс дорівнює x, позначимо останній елемент цього префікса як небезпечний.

- Якщо цей елемент не є найбільшим додатним числом, поміняємо його з найбільшим.
- Якщо він уже дорівнює найбільшому числу, перенесемо найменше додатне число в кінець.

Після цього жоден префікс серед додатних чисел не дорівнює x.

Потім дописуємо всі недодатні числа в кінець. Після всіх додатних чисел поточна сума більша за x, а після додавання всіх недодатних чисел вона стане рівною S, де $S > x$. Отже, під час додавання недодатних чисел префіксна сума теж не стане рівною x.

Випадок 2: $mn = mx$

Тобто всі додатні числа однакові. Позначимо це значення як c.

Якщо x не ділиться на c, то можна поставити всі додатні числа на початок. Префіксні суми, які складаються тільки з додатних чисел, будуть кратні c, тому не дорівнюватимуть x.

Після цього додаються недодатні числа. Поточна сума зменшується, але фінальна сума дорівнює S, а $S > x$. Тому на цій частині префіксна сума також не стане рівною x.

Тепер нехай x ділиться на c.

Якщо існує недодатне число y, яке не ділиться на c, поставимо його першим. Після нього поставимо всі додатні числа, а потім усі інші недодатні.

Поки ми додаємо додатні числа, префіксна сума має такий самий залишок за модулем c, як і y. Оскільки y не ділиться на c, ця сума не може дорівнювати x, бо x ділиться на c.

Після всіх додатних чисел поточна сума вже більша за x, а після додавання решти недодатних чисел вона стане рівною S, де $S > x$. Отже, і тут небезпечного префікса не буде.

Залишається останній випадок: x ділиться на c, усі додатні числа дорівнюють c, і всі недодатні числа також діляться на c.

Тоді всі числа можна поділити на c. Після цього всі додатні числа стануть рівними 1, а x стане додатним цілим числом.

Оскільки фінальна сума більша за x, у будь-якій перестановці префіксна сума колись стане більшою за x. Уперше це може статися тільки після додавання числа 1. Отже, безпосередньо перед цим префіксна сума була рівною x.

Тому в цьому випадку відповідь "NO".

Складність

У розв'язку потрібно лише відсортувати масив та пройтися по ньому кілька разів.

Загальна складність — $\mathcal{O}(n \log n)$ на тестовий випадок.

Задача C. Відгадай число

Автор задачі: **Антон Тригуб**
Задачу підготував: **Костянтин Денисов**
Розбір написав: **Костянтин Денисов**

Блок 1

У першому блоці $n \leqslant 10$. Оскільки за умовою n не ділиться на 10, то можливі лише значення $1, 2, \ldots, 9$.

Достатньо зробити один запит з $x = 1$. У відповідь ми отримаємо першу цифру числа n, тобто саме число n.

Блоки 2–4

У цих блоках можна використати перебір кандидатів. Спочатку випишемо всі можливі значення n у межах обмеження блока, які не діляться на 10.

Після кожного запиту x будемо отримувати цифру d і залишати лише ті кандидати v, для яких перша цифра числа $v \cdot x$ дорівнює d. Якщо після кількох запитів залишився один кандидат, то це і є відповідь.

Щоб запитів було менше, можна кожного разу вибирати x, який найкраще розділяє поточну множину кандидатів. Для цього достатньо перебрати всі значення x від 1 до певної невеликої межі або просто багато випадкових значень x і вибрати таке, для якого найбільша група кандидатів з однаковою відповіддю буде мінімальною.

Оскільки в блоках 2–4 кандидатів небагато, такого підходу достатньо, щоб знайти n за дозволену кількість запитів.

Блок 5

У цьому блоці n є повним квадратом. Тут також можна використати перебір кандидатів.

Якщо $n = y^2$, то з умови $n < 10^8$ маємо $y < 10^4$. Отже, існує лише $O(10^4)$ можливих кандидатів. Залишимо серед них тільки ті квадрати, які не діляться на 10.

Далі діємо так само: робимо запити, а після кожного запиту відкидаємо всі квадрати, які не могли дати отриману відповідь.

Залишається зрозуміти, як вибирати хороший наступний запит. Перебирати всі x від 1 до 10^9 занадто довго, але це й не потрібно.

Для фіксованого кандидата v відповідь на запит x змінюється тільки тоді, коли число $v \cdot x$ переходить через число вигляду

$$d \cdot 10^p,$$

де $1 \leqslant d \leqslant 9$. Тому множина кандидатів розбивається по відповідях однаково на всіх проміжках між такими точками.

Отже, достатньо розглядати лише значення x, близькі до

$$\left\lceil \frac{d \cdot 10^p}{v} \right\rceil$$

для всіх поточних кандидатів v, усіх цифр d і всіх можливих степенів p, для яких $1 \leqslant x \leqslant 10^9$.

Для кожного такого x порахуємо, скільки кандидатів дадуть відповідь 1, скільки — відповідь 2, і так далі. Виберемо запит, для якого найбільша з цих груп є мінімальною.

Кандидатів лише $O(10^4)$, а для кожного кандидата є лише $O(\log 10^9)$ важливих меж. Тому такий підбір запитів достатньо швидкий для цього блока.

Загальна ідея

Для наступних блоків використаємо інший підхід. Візьмемо

$$M = 10^8.$$

Зробимо запит з $x = M$ і нехай відповідь дорівнює s. Оскільки $n < 10^8$, множення на 10^8 просто дописує до числа n вісім нулів справа. Тому перша цифра числа $n \cdot M$ дорівнює першій цифрі самого

n.

Тепер будемо шукати перше значення r, для якого перша цифра числа $n \cdot r$ вже не дорівнює s.

Таке значення r достатньо шукати на відрізку

$$[M, 2M].$$

Справді, якщо число nM починається з цифри s, то наступна межа, на якій перша цифра може змінитися, має вигляд

$$(s+1) \cdot 10^p$$

для деякого p. Ця межа знаходиться не далі ніж у два рази від поточного значення nM, тому до $x = 2M$ перша цифра точно зміниться.

Значення r можна знайти бінарним пошуком. Якщо для деякого x відповідь на запит дорівнює s, то шукане r більше за x. Інакше $r \leqslant x$.

Після того як знайшли r, потрібно відновити n. Нехай T — межа, на якій відбулася зміна першої цифри. Тоді

$$T = (s+1) \cdot 10^p$$

для деякого p, і r є найменшим цілим числом, для якого $nr \geqslant T$. Отже,

$$r = \left\lceil \frac{T}{n} \right\rceil.$$

Тепер уявімо, що ми дописали до числа n кілька нулів справа так, щоб отримане число мало ту саму довжину, що й M. Позначимо це число через

$$N = n \cdot 10^q.$$

Оскільки n не ділиться на 10, після знаходження N ми зможемо просто видалити всі нулі в кінці і отримати початкове число n.

Якщо одночасно домножити межу T на 10^q, то отримаємо

$$T' = T \cdot 10^q.$$

Тоді

$$r = \left\lceil \frac{T'}{N} \right\rceil.$$

Для такого вибору N можна відновити його як

$$N = \left\lceil \frac{T'}{r} \right\rceil.$$

Ми не знаємо точне значення T', але знаємо, що воно має вигляд

$$s+1, \quad (s+1) \cdot 10, \quad (s+1) \cdot 10^2, \quad \ldots$$

Тому будемо перебирати такі значення T' і брати перше, для якого

$$\left\lceil \frac{T'}{r} \right\rceil > 10^7.$$

Отримане число буде дорівнювати n з, можливо, кількома дописаними нулями справа. Після видалення всіх кінцевих нулів отримаємо відповідь.

Блоки 6–10

Для блоків 6–10 достатньо застосувати описану загальну ідею.

Ми робимо один початковий запит з $x = 10^8$, а потім бінарним пошуком шукаємо першу зміну відповіді на відрізку

$$[10^8, 2 \cdot 10^8].$$

Кількість запитів дорівнює

$$1 + \lceil \log_2 10^8 \rceil = 28.$$

Отже, алгоритм вкладається в обмеження на кількість запитів і проходить усі блоки.

Залишається пояснити, чому відновлення відповіді в кінці є коректним. Нехай r — перше число, для якого перша цифра числа nr змінилася. Тоді для деякого p межа має вигляд

$$T = (s+1) \cdot 10^p,$$

і виконується

$$(r-1)n < T \leqslant rn.$$

Тепер домножимо n і T на однаковий степінь десятки. Отримаємо числа

$$N = n \cdot 10^q, \qquad T' = T \cdot 10^q.$$

Для них усе ще виконується

$$(r-1)N < T' \leqslant rN.$$

Звідси

$$\left\lceil \frac{T'}{r} \right\rceil \leqslant N.$$

З іншого боку, ми вибираємо T' так, щоб отримане число було більше за 10^7. Тому N має ту саму довжину, що й число $M = 10^8$, з точністю до можливих кінцевих нулів. У цьому випадку округлення вже однозначно відновлює саме N, тобто

$$\left\lceil \frac{T'}{r} \right\rceil = N.$$

Отже, ми знаходимо число N, яке є початковим числом n з деякою кількістю дописаних нулів справа. Оскільки за умовою n не ділиться на 10, після видалення всіх кінцевих нулів залишиться саме n.

Задача D. mex plus

Автор задачі: **Костянтин Денисов**
Задачу підготував: **Андрій Столітній**
Розбір написав: **Костянтин Денисов**

Блок 1

У цьому блоці $n \leqslant 20$ і немає запитів. Можна перебрати всі способи вибору чисел з пар.

Для кожної пари є два варіанти: перше число піде в A, а друге в B, або навпаки. Отже, всього є 2^n варіантів. Для кожного з них будуємо масиви A та B, рахуємо їхні mex і оновлюємо відповідь.

Складність такого розв'язку становить $O(2^n \cdot n)$, чого достатньо для цього блока.

Блок 2

У цьому блоці всі пари мають вигляд $(x, 10^9)$. Значення 10^9 ніколи не впливає на mex, бо mex не може бути таким великим: у кожному масиві лише n елементів.

Тому кожна пара фактично дає нам одну копію числа x, яку можна покласти або в A, або в B.

Нехай cnt_i — кількість пар, у яких перше число дорівнює i. Щоб число i було і в A, і в B, потрібно мати хоча б дві копії i. Щоб воно було хоча б в одному з масивів, достатньо однієї копії.

Позначимо через p найменше число, для якого $cnt_p = 0$, а через q — найменше число, для якого $cnt_q < 2$. Тоді можна зробити mex одного масиву рівним p, а mex іншого — q. Кращої відповіді бути не може, бо для більшого mex одному масиву потрібна хоча б одна копія кожного малого числа, а двом масивам одночасно — хоча б дві копії.

Отже, відповідь дорівнює $p+q$. Після кожного запиту достатньо оновити одну кількість cnt_x і заново знайти ці два mex. Це можна робити за допомогою множин або просто перебором малих значень.

Блок 3

У цьому блоці всі пари мають вигляд $(0, y)$. Для додатних чисел ситуація схожа на блок 2: кожна пара $(0, y)$ дає одну копію числа y, яку можна покласти або в A, або в B.

Основна відмінність полягає в числі 0. Якщо ми кладемо y в A, то 0 потрапляє в B, і навпаки. Тому для того, щоб обидва mex були додатними, потрібно зробити так, щоб 0 потрапив в обидва масиви.

Якщо є пара $(0, 0)$, то число 0 одразу потрапляє в обидва масиви. Також якщо для деякого $y > 0$ є хоча б дві пари $(0, y)$, то одну з них можна використати, щоб покласти y в A і 0 в B, а іншу — щоб покласти 0 в A і y в B. У такому випадку число 0 вже є в обох масивах, а додатні числа можна розподіляти так само, як у блоці 2.

Тобто, якщо cnt_i — кількість пар $(0, i)$ для $i > 0$, то шукаємо найменше додатне p, для якого $cnt_p = 0$, і найменше додатне q, для якого $cnt_q < 2$. Відповідь у цьому випадку дорівнює $p + q$.

Залишився випадок, коли немає пари $(0, 0)$ і жодне додатне число не зустрічається хоча б двічі. Тоді кожне додатне число можна покласти лише в один із масивів. Щоб зробити 0 присутнім в обох масивах, потрібно хоча б одну пару орієнтувати в один бік і хоча б одну — в інший. Тобто одним додатним числом доведеться пожертвувати для одного з масивів.

У такому випадку оптимально віддати одному масиву якомога довший префікс додатних чисел $1, 2, 3, \ldots$, а в інший масив покласти одне число, яке найменше шкодить цьому префіксу. Інакше кажучи, якщо є число після першого пропущеного додатного, можна пожертвувати ним. Якщо ж усі наявні додатні числа утворюють префікс, то найкраще пожертвувати найбільшим із них.

Після кожного запиту змінюється лише одна кількість cnt_i, тому потрібні величини можна підтри-

мувати множинами або перераховувати напряму для цієї спеціальної структури.

Блок 4

У цьому блоці $n \leqslant 500$ і $q \leqslant 500$. Можна після кожного запиту розв'язувати задачу з нуля.

Побудуємо граф, вершинами якого є числа, а кожна пара (x, y) є ребром між вершинами x та y. Далі для поточного графа застосуємо розв'язок для випадку без запитів, описаний у наступному блоці.

Оскільки розмір графа малий, навіть повна перебудова після кожного запиту достатньо швидка.

Блок 5

Розглянемо задачу без запитів.

Побудуємо граф: вершинами будуть числа, а кожна пара (x, y) задає ребро між x та y. Ребра можуть повторюватися, також можуть бути петлі.

Тепер будемо орієнтувати ребра. Якщо ребро (x, y) орієнтоване з x до y, то x потрапляє в масив A, а y — у масив B.

Отже, число v входить у масив A тоді і тільки тоді, коли з вершини v виходить хоча б одне ребро. Аналогічно, v входить у масив B тоді і тільки тоді, коли у вершину v входить хоча б одне ребро.

Тому нам потрібно орієнтувати ребра так, щоб найменша вершина без вихідних ребер і найменша вершина без вхідних ребер були якомога більшими.

Розглянемо одну компоненту зв'язності.

Якщо компонента містить цикл, то можна орієнтувати ребра циклу по колу. Тоді всі вершини на циклі матимуть і вхідне, і вихідне ребро. Усі дерева, які підвішені до циклу, можна орієнтувати від листків до циклу. Тоді кожна вершина компоненти матиме вихідне ребро, а без вхідного ребра можуть залишитися тільки листки.

Якщо компонента є деревом, то виберемо в ній найбільшу вершину і орієнтуємо всі ребра в її сторону. Тоді всі вершини, крім цієї найбільшої, матимуть вихідне ребро. З іншого боку, без вхідного ребра залишаться листки дерева.

Звідси отримуємо відповідь. Значення $\operatorname{mex}(B)$ дорівнює найменшому листку в усьому графі. Значення $\operatorname{mex}(A)$ дорівнює мінімуму серед максимумів усіх компонент, які є деревами.

Тут також треба пам'ятати про числа, яких взагалі немає серед пар. Таке число можна вважати окремою компонентою з однієї вершини. Тому достатньо розглядати лише числа до $n+1$, бо mex масиву довжини n не може бути більшим за n.

Отже, для блока без запитів можна побудувати граф, знайти компоненти зв'язності, для кожної компоненти визначити, чи є вона деревом, її максимум, а також переглянути степені вершин для пошуку найменшого листка.

Складність становить $O(n \log n)$ або $O(n)$ залежно від реалізації стискання координат.

Блок 6

У цьому блоці всі запити мають вигляд додавання пари.

Оскільки ребра тільки додаються, можна підтримувати компоненти за допомогою DSU. Для кожної компоненти будемо зберігати її найбільшу вершину і інформацію про те, чи є в ній цикл.

Також потрібно підтримувати дві множини:

- вершини, які зараз є листками;
- значення максимумів тих компонент, які зараз є деревами.

Тоді відповідь після кожного запиту дорівнює сумі мінімального елемента першої множини і мінімального елемента другої множини.

При додаванні ребра змінюються степені двох його кінців і, можливо, об'єднуються дві компоненти. Усі ці зміни можна оновити в DSU і в множинах за $O(\log n)$.

Блок 7

У цьому блоці всі запити мають вигляд видалення пари.

Такий випадок можна обробити у зворотному порядку. Спочатку візьмемо граф після всіх видалень. Далі будемо йти по запитах з кінця до початку. Видалення пари у прямому порядку перетворюється на додавання цієї пари у зворотному порядку.

Отже, ми знову отримуємо задачу тільки з додаваннями ребер, яку можна розв'язати DSU, як у попередньому блоці. Відповіді потрібно буде записати і потім вивести у правильному порядку.

Блок 8

У цьому блоці структура графа в кожен момент часу суттєво обмежена: якщо дві пари мають спільне число з третьою, то в усіх трьох пар є спільне число.

Інтуїтивно це означає, що кожна нетривіальна компонента поводиться як зірка: у її ребер є спільна центральна вершина. Тому для компоненти легко підтримувати її листки, максимум і наявність циклу, не використовуючи повний алгоритм динамічної зв'язності.

Після кожного додавання або видалення достатньо оновити локальну інформацію про відповідну зірку і перерахувати внесок цієї компоненти у дві глобальні множини. Відповідь, як і раніше, є сумою найменшого листка та найменшого максимуму серед деревних компонент.

Блоки 9, 10: повне рішення

Для повного розв'язку використаємо офлайн-динамічну зв'язність.

Для кожної пари знайдемо проміжки часу, на яких вона присутня в наборі. Якщо пара була додана в момент l і видалена в момент r, то вона активна на відрізку $[l, r)$. Початкові пари активні з моменту 0.

Побудуємо дерево відрізків по часу. Кожне ребро додамо у всі вершини дерева відрізків, які повністю покривають його проміжок активності.

Після цього обійдемо дерево відрізків DFS-ом. Коли заходимо у вершину дерева відрізків, додаємо в DSU всі ребра, записані в цій вершині. Коли виходимо з неї, відкочуємо DSU до попереднього стану.

Для цього потрібен DSU з відкатами. Він має підтримувати:

- об'єднання компонент;
- додавання ребра всередині однієї компоненти;
- зміну степенів кінців ребра;
- відкат усіх змін.

Також під час цих змін підтримуємо дві глобальні множини:

- усі вершини, які зараз є листками;
- максимуми компонент, які зараз є деревами.

Якщо ми знаходимося в листку дерева відрізків, то всі ребра, активні в цей момент часу, вже додані в DSU. Тому поточна відповідь дорівнює

$\min(\text{листки}) + \min(\text{максимуми деревних компонент})$.

Кожне ребро додається в $O(\log q)$ вершин дерева відрізків. Кожне додавання або відкат у DSU коштує $O(\log n)$ через оновлення множин. Тому сумарна складність становить $O((n+q)\log^2(n+q))$.

Задача E. Підмножини дерева

Автор задачі: Костянтин Денисов
Задачу підготував: Костянтин Денисов
Розбір написав: Едуард Тихонюк

Блок 1

Для першого блока можна використати повний перебір. Оскільки $n \leqslant 20$, переберемо всі підмножини вершин, перевіримо умову з задачі і для кожного розміру k запам'ятаємо, які xor-и були отримані.

Перевірку підмножини можна робити напряму: для кожної вибраної вершини v перебрати її дітей і перевірити, що вони також вибрані. Сумарно за одну перевірку ми перебираємо ребра дерева, тому це працює за $O(n)$. Після цього рахуємо розмір множини і xor її значень. Сумарна складність буде $O(2^n \cdot n)$.

Блок 2

У цьому блоці всі вершини, крім кореня, є листками. Отже, будь-яка допустима множина або не містить корінь, або містить усе дерево. Якщо корінь не вибрано, то можна вибирати довільну підмножину листків.

Для $1 \leqslant k \leqslant n-2$ відповідь легко визначити за значеннями листків. Нехай серед листків є c_0 нулів і c_1 одиниць. Якщо ми вибираємо k листків, то кількість вибраних одиниць може бути будь-якою в проміжку

$$\max(0, k - c_0) \leqslant r \leqslant \min(k, c_1).$$

Отриманий xor дорівнює парності r. Тому достатньо перевірити, чи в цьому проміжку є числа обох парностей. Розміри $k = n-1$ і $k = n$ потрібно перевірити окремо: для $n-1$ можна вибрати всі листки, а для n можна вибрати лише все дерево разом з коренем. Порахувати c_0, c_1 і пройти всі розміри k можна за $O(n)$.

Загальна ідея

Для наступних блоків будемо рахувати динаміку по піддеревах. Для кожної вершини v будемо зберігати масив dp_v, де $dp_v[t]$ є відповіддю на задачу для піддерева v при $k = t$.

Порожню множину у кожному піддереві завжди можна вибрати, тому $dp_v[0] = 0$. Після того як ми об'єднали інформацію з усіх синів вершини v, потрібно врахувати саму вершину v. Якщо з дітей можна було вибрати $sz(v) - 1$ вершин з xor x, то записуємо $dp_v[sz(v)] = x \oplus a_v$.

Блок 3

Спочатку розглянемо випадок, коли у вершини рівно два сини. Тоді треба об'єднати дві вже пораховані динаміки. Для цього блока достатньо перебрати всі переходи, для яких $i + j = t$, $0 \leqslant i, j$ та $0 \leqslant t \leqslant n$:

$$C[t] = \bigcup_{\substack{i+j=t \\ 0 \leqslant i,j}} A[i] \oplus B[j].$$

Тут значення 0 або 1 означає єдиний можливий xor, а значення 2 означає, що можливі обидва. Якщо для одного розміру t з'являються і xor 0, і xor 1, то записуємо $C[t] = 2$.

Якщо у вершини більше двох синів, будемо зливати їх по одному. Спочатку беремо порожню динаміку з $dp[0] = 0$, потім об'єднуємо її з динамікою першого сина, результат об'єднуємо з динамікою другого сина і так далі. Після того як усі сини об'єднані, додаємо саму вершину: зі стану розміру $sz(v) - 1$ отримуємо стан $sz(v)$.

Таких переходів для одного злиття $O(n^2)$. Кожне злиття відбувається в батьку якоїсь вершини, тому сумарно злиттів буде не більше, ніж неунікальних батьків вершин, тобто $n - 1$. Отже, загальна складність $O(n^3)$, що проходить обмеження блока.

Блок 4

Нехай m — це розмір A, а k — це розмір B. Тоді під час злиття перебираємо $0 \leqslant i \leqslant m$ та $0 \leqslant j \leqslant k$:

$$C[t] = \bigcup_{\substack{i+j=t \\ 0 \leqslant i \leqslant m \\ 0 \leqslant j \leqslant k}} A[i] \oplus B[j].$$

Одне злиття працює за $O(mk)$. Сумарно це можна оцінити через LCA: кожна пара вершин враховується тільки під час об'єднання в їхньому LCA, тому загальна кількість переходів по всіх вершинах дорівнює $O(n^2)$. Далі для вершини v послідовно зливаємо динаміки всіх її синів, а потім додаємо можливість вибрати саму вершину v.

Об'єднання за лінію

Щоб прискорити злиття двох масивів A і B, нехай $|A| = m$, $|B| = k$. Для кожного t завжди можна взяти один кандидат, наприклад крайню доступну пару індексів з $i + j = t$. Після цього треба лише зрозуміти, чи існує інша пара індексів, яка дає протилежний xor. Якщо існує, відповідь для цього t дорівнює 2, інакше вона дорівнює xor цього єдиного кандидата.

Для фіксованого t усі допустимі пари індексів мають вигляд $i \in [l_1, r_1]$, $j = t - i \in [l_2, r_2]$. Якщо серед потрібних значень є $A[i] = 2$ або $B[j] = 2$, то відповідь для цього t одразу дорівнює 2. Інакше всі значення на потрібних відрізках дорівнюють 0 або 1, і можна хешами порівняти відрізок $A[l_1..r_1]$ з розвернутим відрізком $B[l_2..r_2]$, можливо інвертувавши усі значення одного з них. Тоді всі значення $C[t]$ для злиття двох масивів рахуються за $O(m + k)$.

Блок 5

У цьому блоці сумарна кількість листків за всіма тестами не перевищує 100. Якщо стиснути кожен шлях без розгалужень, то залишиться дерево розміру $O(L)$, де L — кількість листків. На цьому стисненому дереві можна виконувати ті самі злиття, а довгі шляхи обробляти напряму. Оскільки для кожного злиття $m, k \leqslant n$, сумарно отримаємо $O(nL)$.

Блок 6

Після видалення вершини 1 кожна компонента є бамбуком, і таких компонент не більше двох. Динаміка для одного бамбука рахується напряму, а в корені треба лише об'єднати динаміки цих компонент. Оскільки таке злиття працює за $O(m + k)$, у цьому блоці отримуємо $O(n)$.

Блок 7

Після видалення вершини 1 кожна компонента є бамбуком, але компонент може бути багато. Порахуємо динаміку для кожного бамбука, а потім будемо щоразу зливати два найменші поточні масиви. Нехай деякий елемент зараз лежить у масиві розміру s, і цей масив вибрали для злиття. Інший вибраний масив має розмір хоча б s, бо наш масив був одним із двох найменших. Отже, після злиття елемент лежить у масиві розміру хоча б $2s$. Якщо елемент брав би участь у d злиттях, то розмір його масиву

був би хоча б 2^d. Але розмір масиву не може перевищити n, тому $2^d \leqslant n$, а отже $d \leqslant \log_2 n$. Сумарна складність буде $O(n \log n)$.

Блок 8

Дерево є повним бінарним. Якщо зливати динаміки дітей у кожній вершині об'єднанням за лінію, то рівні дерева поводяться як merge sort: на одному рівні сумарні розміри всіх злиттів дорівнюють $O(n)$, а рівнів $O(\log n)$. Отже, отримуємо $O(n \log n)$.

Блоки 9, 10: повне рішення

Залишається зробити злиття достатньо швидким для довільного дерева. Нехай зливаємо великий масив A розміру m і менший масив B розміру k, $m \geqslant k$. Перші k і останні k значень нового масиву можна порахувати за $O(k)$: для цього застосовуємо злиття за лінію окремо до префікса і суфікса.

Потрібно швидко порахувати середину $C[k+1..m]$. Для кожної такої позиції є кандидат з A, бо можна взяти $B[0]$. Треба лише перевірити, чи існує інший кандидат, який змінює відповідь на 2.

Якщо у масиві B є значення 2 на якійсь позиції $j \geqslant 1$, то для кожної середньої позиції i, де $A[i-j]$ існує, можна взяти пару $A[i-j]$ і $B[j]$. Оскільки $B[j] = 2$, ця пара дає обидва хог-и, тому $C[i]$ одразу стає 2.

Складний випадок виникає тоді, коли в B немає значень 2. Тоді всі його значення є лише 0 або 1. Якщо для двох сусідніх позицій середини кандидат з A лишається єдиним, з рівностей

$$A[i] = A[i-1] \oplus B[1], \qquad A[i+1] = A[i] \oplus B[1]$$

випливає періодичність $A[i+1] = A[i-1]$. Аналогічно отримуємо, що B мусить мати вигляд

$$B[2q] = 0, \qquad B[2q+1] = c$$

для деякого $c \in \{0, 1\}$. Тобто при $c = 0$ маємо форму

$$0,\ 0,\ 0,\ 0,\ \ldots$$

а при $c = 1$ маємо форму

$$0,\ 1,\ 0,\ 1,\ \ldots$$

Після цього масив A розбивається на блоки вигляду

$$s,\ s \oplus c,\ s,\ s \oplus c,\ \ldots$$

Є п'ять типів таких блоків:

$$0,\ 0,\ 0,\ 0,\ \ldots$$
$$0,\ 1,\ 0,\ 1,\ \ldots$$
$$1,\ 1,\ 1,\ 1,\ \ldots$$
$$1,\ 0,\ 1,\ 0,\ \ldots$$

Окремий випадок:

$$2,\ 2,\ 2,\ 2,\ \ldots$$

Достатньо підтримувати такі блоки у стисненому вигляді. Далі розглянемо злиття з періодичним масивом B з параметром c.

Якщо параметр c блока збігається з параметром c у масиві B, то змінитися може тільки префікс довжини $k-1$. Для перших $k-1$ позицій частина індексів $i-j$ виходить за межі поточного блока і потрапляє в інший блок, де може бути інший тип періоду. Через це узгодженість кандидатів може зламатися: перші $k-2$ позиції стають 2, а позицію $k-1$ треба перевірити окремо.

Для позиції, що лежить хоча б на $k-1$ елементів правіше від початку такого блока, усі індекси $i-j$ для $0 \leqslant j \leqslant k$ залишаються всередині цього ж блока. Там значення йдуть з тим самим періодом, що й B, тому всі кандидати узгоджуються між собою і відповідь не змінюється.

Якщо параметр c блока відрізняється від параметра c у масиві B, то весь цей блок стає 2, можливо окрім першого елемента. Для кожної позиції, починаючи з другої, можна взяти два сусідні кандидати: через $B[0]$ і через $B[1]$. Оскільки періоди в A і B різні, ці два кандидати дають різні хог-и, тому відповідь стає 2.

Якщо ж B має іншу форму, тобто не є одним з описаних періодичних масивів, то на середньому відрізку не можуть лишитися дві сусідні позиції без значення 2. Якщо дві сусідні позиції не змінюються, то це вже був би розібраний періодичний випадок для B. Якщо ж зліва від позиції стоїть 2, то ця позиція теж стає 2 через перехід $A[i-1] \oplus B[1]$. Інакше, якщо зліва не стоїть 2, то маємо дві сусідні позиції без значення 2, а такий випадок уже розглянули. Крайні позиції, для яких немає такого лівого сусіда, перевіряємо окремо. Отже, серед кожних двох сусідніх позицій хоча б одна стає 2. Тоді в лоб будемо перевіряти тільки ті позиції, де $A[i] \neq 2$.

Амортизовано це працює швидко. Коли ми робимо прохід, кількість позицій зі значенням 2 збільшується пропорційно кількості перевірених позицій. Значення 2 ніколи не переходить назад у 0 або 1. Отже, кожна позиція може стати 2 лише один раз, тому сумарно в усіх таких проходах ми опрацюємо $O(n)$ елементів.

Отже, повне рішення зливає динаміки дітей у сумарно $O(n \log n)$ або $O(n \log^2 n)$ залежно від реалізації підтримки блоків і проходить блоки 9 та 10.