Задача A1. Краківські пам'ятки

Автор: **Павло Ціцей**
Підготовка задачі: **Павло Ціцей**
Розбір: **Павло Ціцей**

Блок 1: $x_i^2 + y_i^2 \leqslant 18$ для всіх i:

Ми можемо замінити (x, y) на (a, b), якщо $x^2 + y^2 = a^2 + b^2$, що означає, що ми можемо замінити $(-x, y)$ на (x, y) та (x, y) на (y, x), тому ми можемо зробити всі координати у вхідних даних типу $0 \leqslant x \leqslant y$ і згрупувати їх разом. Таким чином, у нас є 10 типів таких груп (якщо не враховувати $(0, 0)$). Отже, ми можемо вирішити це як третій блок.

Блок 2: $n = 1$:

Зверніть увагу, що відстань від $(0, 0)$ до (x, y) дорівнює $\sqrt{x^2 + y^2}$, тому якщо ми змінюємо (x, y) на (a, b), де $x^2 + y^2 = a^2 + b^2$, тоді відстань не змінюється. Отже, відповідь це сума квадратів чисел, помножена на 4.

Блок 3: $n \leqslant 10$:

Ми можемо спробувати кожну перестановку точок, яка визначає шлях, яким ми йдемо, а перший і останній елемент будуть $(0, 0)$. Загальна складність буде $O(n!)$.

Блок 4: без додаткових обмежень:

Ми можемо змінити координати кожної точки з (x, y) на $(0, \sqrt{x^2 + y^2})$. Тоді відповідь буде максимальним з цих точок, помноженим на 2 (оскільки нам потрібно повернутися).

Задача A2. Сакурако та дивна перестановка

Автор: **Богдан Фейса**
Підготовка завдання: **Богдан Фейса**
Розбір: **Богдан Фейса**

Блок 1: $n \leqslant 8$:

У цьому рішенні блоку ми спробуємо згенерувати всі можливі списки перестановок і перевірити, чи підходять вони для нашої перестановки t. Часова складність $O(\binom{n \cdot (n-1)}{8} \cdot n)$. Що менше ніж $2.5 \cdot 10^8$.

Блок 2: $n \leqslant 100$:

Для нашої перестановки ми створимо її циклічну декомпозицію наступним чином:

1. створити її графічне представлення
2. обчислити для кожної вершини множину тих, що доступні з неї за $O(n^2)$.
3. знайти порядок тих чисел у циклі.

Тепер, коли ми маємо циклічну декомпозицію нашої перестановки, ми беремо її транспозиції, і це наша відповідь. Часова складність $O(n^2)$.

Блок 2: $n \leqslant 10^5$:

У цьому рішенні блоку ми можемо використовувати $O(n \log n)$ час для циклічної декомпозиції перестановки. Коли ми маємо всі цикли, ми записуємо ребра циклів, і це наша відповідь. Тому що, використовуючи ці перестановки, ми створюємо обернену до t. І $t[t^{-1}[i]] = i$, і це те, що нам потрібно знайти.

Блок 2: повне рішення:

Ми змінюємо наш код, щоб розбити перестановку на цикли за лінійний час. Потім ми записуємо ребра циклу, і це наша відповідь. Часова складність $O(n)$.

Задача B1. UJGOI у Кракові

Автор: **Олександр Тимкович**
Підготовка задачі: **Олександр Тимкович, Павло Ціцей**
Розбір: **Олександр Тимкович**

Блок 1: $a_i \geqslant 1$:

Основне спостереження полягає в тому, що масив не має відрізка довжини $\ell \in \{2, 3, \ldots, n\}$ з мажоруючим елементом тоді і тільки тоді, коли масив не має відрізка довжини $\ell \in \{2, 3\}$ з мажоруючим елементом.

Блок 2: $a_i = 0$:

Для $k \geqslant 3$ одна з можливих відповідей - $a = [1, 2, 3, 1, 2, 3, 1, 2, 3, \ldots]$. Для $k < 3$ відповідь майже завжди -1, за винятком достатньо малих n, де попередня конструкція також працює.

Блок 3: немає $i > 1$, так що $a_i = a_{i-1} = 0$:

Для кожного невідомого елемента ми знаємо обох сусідів (якщо вони існують). Рішення подібне до $k > 5$ з деякими крайніми випадками.

Блок 4: $k > 5$:

Для $k > 5$ ми можемо використовувати жадібний підхід, замінюючи кожен невідомий елемент на $\texttt{mex}\{a_{i-2}, a_{i-1}, a_i, a_{i+1}, a_{i+2}\}$.

Блок 5: $k \leqslant 5$:

Для $k \leqslant 5$ підхід автора включав динамічне програмування, де деякі стани означали останні три значення, що вирішують підзадачу на масиві $[a_1, \ldots, a_i]$.

Блок 6: без додаткових обмежень:

Поєднайте рішення блоків 4 і 5.

Задача B2. UJGOI Таблиця результатів

Автор: **Павло Ціцей**
Підготовка задачі: **Павло Ціцей**
Розбір: **Павло Ціцей**

Блок 1: a_i непарне для всіх i, $n \leqslant 1000$:

Давайте зробимо префіксну суму масиву, щоб ми могли знайти суму будь-якого підмасиву за $O(1)$, а потім пройти через всі дійсні інтервали. g_{odd} цього інтервалу не дорівнює нулю, якщо кількість непарних елементів непарна, тобто довжина інтервалу непарна. Отже, ми можемо просто перевірити для кожного інтервалу, чи довжина непарна, і якщо так, додати його суму до відповіді.

Блок 2: $n \leqslant 1000$:

Тут нам потрібно створити два додаткові масиви. Нехай це буде масив odd і $even$, де $odd_i = 1$, якщо a_i непарне, а в іншому випадку 0. Для масиву $even$, $even_i = 1$, якщо a_i парне, а в іншому випадку 0. Ми можемо зробити префіксну суму на цьому масиві, а потім нам потрібно додати $g_{odd}(l, r)$ для деякого дійсного інтервалу l, r, якщо кількість i, для яких odd_i дорівнює 1 (тобто сума елементів на підмасиві масиву odd) непарна, що можна обчислити за допомогою префіксної суми. Те ж саме стосується парних, тому ми можемо просто пройти через всі інтервали.

Блок 3: a_i непарне для всіх i:

Давайте зафіксуємо якийсь елемент i і підрахуємо, скільки разів він включається в суму на інтервалі. Як було зазначено, a_i включається в g_{odd}, якщо кількість непарних елементів на інтервалі непарна, тобто якщо довжина інтервалу непарна, що означає, що l і r мають різну парність. Отже, ми можемо знайти це, просто підрахувавши кількість r в діапазоні від i до n, які є непарними і парними, і те ж саме для l, але в діапазоні від 1 до i.

Блок 4: без додаткових обмежень:

Давайте узагальнимо рішення для третього блоку. Отже, якщо елемент непарний, нам потрібно підрахувати кількість інтервалів, де цей елемент включений в g_{odd}, в іншому випадку в g_{even}. Ми припустимо, що завжди включаємо його в g_{odd}, оскільки для g_{even} це симетрично. Отже, для елемента i він включений в $g_{odd}(l, r)$, якщо a_i непарне, а інтервал від l до r має непарну кількість елементів. Давайте створимо масив odd, як у другому блоці, але з ідеєю з третього. Отже, ми фіксуємо деяке i і нам потрібно, щоб парність odd_r відрізнялася від парності odd_l. Для цього ми можемо створити два нові масиви $odd[0]$ і $odd[1]$, де ми підрахуємо кількість парних елементів масиву odd на префіксі довжини i і кількість непарних таких елементів відповідно. Тепер ми можемо підрахувати кількість odd_r, які є парними або непарними, використовуючи префіксну суму на масиві $odd[0]$ або $odd[1]$, і те ж саме для l.

Задача C1. Сакурако та її подорож

Автор: **Богдан Фейса**
Підготовка завдання: **Богдан Фейса**
Розбір: **Богдан Фейса**

Блок 1: $n, m \leqslant 100$:

Для першого блоку нашого завдання ми будемо перебирати кожну можливу відповідь k ($1 \leqslant k \leqslant n$), створювати матрицю $n \times n$ і перевіряти, чи існує шлях від $(1, 1)$ до (n, n), який не містить жодної клітини, що знаходиться ближче ніж на k до будь-яких туристичних пасток. Це призводить до часової складності $O(n \cdot (n^2 + m))$.

Блок 2: $n, m \leqslant 10^3$:

Для другого блоку цього завдання ми збережемо техніку побудови матриці, але замість перебору кожної можливої відповіді k ми будемо використовувати бінарний пошук, щоб знайти найбільше можливе k, таке що існує шлях від $(1, 1)$ до (n, n), який не містить жодної клітини, що знаходиться ближче ніж на k до будь-яких туристичних пасток. Це призводить до часової складності $O(\log n \cdot (n^2 + m))$.

Блок 3: $n \leqslant 10^3$:

Цей блок має таке ж рішення, але тут нам потрібно використовувати bfs від усіх m туристичних пасток, щоб позначити всі клітини, які знаходяться ближче ніж на k до будь-якої туристичної пастки. Це призводить до часової складності $O(\log n \cdot (n^2))$.

Блок 4: або $x_1 = x_2 = \cdots = x_m$ або $y_1 = y_2 = \cdots = y_m$:

Це блок, в якому ми могли б використовувати формулу, щоб спробувати пройти від $(1, 1)$ до (n, n) між кожною парою найближчих туристичних пасток. Часова складність або $O(m \log m + m \log n)$, або $O(m \log m + m)$.

Блок 5: повне рішення:

Ми створимо граф з $m + 4$ вершин. Кожна з перших m вершин позначає туристичну пастку. Інші 4 вершини позначають 4 межі матриці. ($m + 1$ - верхня, $m + 2$ - ліва, $m + 3$ - нижня, $m + 4$ - права)

Ми створимо ребро між парою перших m вершин, якщо немає шляху від $(1, 1)$ до (n, n), який проходить між цими двома туристичними пастками, не проходячи через жодну клітину, відстань до якої від туристичної пастки менша ніж k. Також ми спробуємо з'єднати перші m вершин з іншими 4: якщо ця туристична пастка ближча до межі ніж k, то ми додаємо ребро. Тепер, якщо існує шлях від $m + 1$ до $m + 2$ або від $m + 1$ до $m + 3$ або від $m + 2$ до $m + 4$ або від $m + 3$ до $m + 4$, то немає можливості пройти від $(1, 1)$ до (n, n) з даним k.

Загальна часова складність $O(m^2 \log n)$.

Задача C2. Участь UJGOI

Автор: **Олександр Тимкович**
Підготовка задачі: **Олександр Тимкович, Павло Ціцей**
Розбір: **Олександр Тимкович**

Блок 1: $a_i \leqslant 1$:

Без будь-яких збільшень, XOR завжди не більше одного. Коли ми можемо отримати 0? Розгляньте деякі випадки, коли 0 присутній у масиві, а коли ні.

Блоки $2, 3, 4, 5$:

Зверніться до розв'язання блоку 6. Розв'язки цих блоків все ще існують, але з деякими важливими спостереженнями, які були пропущені для повного розв'язку.

Блок 6: без додаткових обмежень

Можна помітити, що $a_i := a_i + 1$ є такою ж операцією, як $a_i := a_i \oplus \Big(a_i \oplus (a_i + 1)\Big) = a_i \oplus b_i$, де $b_i = a_i \oplus (a_i + 1)$.

А саме, ми замінюємо "+1" на "$\oplus b_i$". У задачі нам потрібно мінімізувати $a_1 \oplus \cdots \oplus a_n$, отже, задача зводиться до знаходження множини $\mathcal{S} \subseteq [n]$, для якої $x = \bigoplus_{i \in \mathcal{S}} b_i$ значення $a_1 \oplus \cdots \oplus a_n \oplus x$ є мінімально можливим.

Наступне спостереження полягає в тому, що кожен b_i має форму $2^k - 1$ для деякого k.

Щоб знайти множину $\mathcal{S}$, ми можемо використовувати жадібний підхід, намагаючись скасувати найбільш значущі біти $A = a_1 \oplus \cdots \oplus a_n$, використовуючи елементи з b. Зверніть увагу, що b має не більше $\lceil \log_2 10^9 \rceil \approx 30$ різних значень, тому для кожного біта з A ми можемо перебрати всі різні елементи в b, щоб знайти відповідне значення.

Задача D1. Олімпіада UJGOI

Автор: **Павло Ціцей**
Підготовка завдання: **Павло Ціцей, Олександр Тимкович**
Розбір: **Павло Ціцей**

Твердження:

Якщо ми маємо два однакових запити, ми можемо працювати з ними як з одним запитом. Це пов'язано з операцією побітового XOR. Маємо, що $a \oplus a = 0$ і $a \oplus b \oplus a = b$. Отже, якщо ми вибираємо дві однакові операції, це еквівалентно вибору жодної з них. У наступних блоках ми будемо вважати, що всі запити унікальні.

Блок 1: $n \leqslant 5$:

Оскільки n до 5, ми можемо мати до 10 запитів, тому можемо просто спробувати всі підмножини запитів, якщо вони правильні.

Блок 2: $q \leqslant 20$:

Ми можемо спробувати всі підмножини запитів. Щоб не проходити масив, ми можемо створити масив b, де $a_i = b_1 \oplus b_2 \oplus \cdots \oplus b_i$ і в цьому масиві запит з l до r еквівалентний зміні лише b_l та b_{r+1}. Отже, ми можемо підрахувати кількість одиниць у масиві b і коли ми змінюємо b_i, якщо воно 1, ми зменшуємо це значення на одиницю і змінюємо b_i, інакше ми збільшуємо значення і змінюємо елемент b_i. Щоб підмножина була відповіддю, нам потрібно, щоб після всіх запитів у нас було нуль одиниць.

Блок 3: $q \leqslant 40$:

Припустимо, що l - це індекс першого елемента 1 в масиві. Нехай $r_1, r_2, ..., r_m$ - це всі запити типу l, r_i у порядку зростання. Тоді, ми використовуємо запит l, r_1 і видаляємо всі інші та додаємо $r_1 + 1, r_i$ для всіх i, де $i > 1$. Таким чином, якщо ми вибираємо запит $r_1 + 1, r_i$, це те ж саме, що вибрати l, r_i. Отже, ми можемо просто робити це багато разів, поки не отримаємо масив, заповнений нулями.

Блок 4: $n \leqslant 64$:

Давайте замінимо наш початковий масив на масив b, як у першому блоці. Зверніть увагу, що його можна представити як двійкове представлення деякого числа x, а запити будуть просто XOR двох степенів двійки ($2^{r+1} \oplus 2^l$). Тоді нам потрібно зробити число рівним 0. Ми можемо спробувати зробити це жадібно. Давайте відсортуємо масив запитів у порядку спадання. Потім ми будемо послідовно створювати новий масив. Ми додаємо найбільший елемент масиву запитів в кінець цього масиву. Для кожного наступного елемента (нехай це буде x), ми проходимо масив, змінюючи x на $min(x, x \oplus c_i)$, де i - це i-й елемент нового масиву. Якщо x не нуль, ми додаємо його в кінець масиву. Щоб отримати відповідь, ми намагаємося змінити n так, як ми змінили x, і нам потрібно зробити його 0.

Блок 5: Усі ℓ_i різні:

Ми можемо використовувати жадібний алгоритм для розв'язання цього. Давайте почнемо з першого входження елемента 1 в масиві. Очевидно, нам потрібно зробити його нульовим, і може бути лише один запит, що починається в цій позиції (нам потрібно взяти той, що починається тут. Інтуїтивно, це тому, що якщо ми змінюємо масив на масив b, ми змінюємо лише 2 елементи, перший і останній). Отже, ми виконуємо цей алгоритм, поки у нас є запити або поки у нас є якісь 1 в масиві.

Блок 6: $a_i = 1$ для всіх i:

Якщо ми створимо масив b, як у другому блоці, він виглядає як $100000\ldots$, і кожен запит змінює два елементи, які ми можемо виразити як зміну індексу, де елемент 1 знаходиться. Отже, ми можемо виразити це як граф, де ми починаємо з вершини 1, а кожне ребро змінює вашу позицію (де ребро - це запит). І вам потрібно дістатися до вершини $n + 1$, що можна зробити за допомогою BFS.

Блок 7: $n,\ q \leqslant 1000$:

Ми створимо масив b, як у другому блоці. Тепер давайте створимо граф, де множина вершин - це вершини масиву b, а множина ребер - це запити. Зверніть увагу, що деяка кількість запитів може утворити шлях, і якщо ми використовуємо всі запити на шляху, то лише елементи, які зміняться, будуть першим і останнім. Отже, щоб вирішити задачу, ми можемо зафіксувати один елемент, який є 1, і знайти шлях до будь-якого іншого елемента, який є 1. Потім беремо всі запити, тобто шлях. Якщо немає іншого елемента, рівного 1, немає способу зробити масив рівним нулю.

Блок 8: без додаткових обмежень:

Ми можемо оптимізувати рішення для блоку 7, створивши MST на графі та зробивши LCA на ньому. Таким чином, ми можемо знайти шлях між двома вершинами за $O(1)$.

Також ми можемо спробувати зробити по-іншому. Ми побудуємо граф, як у блоці 7, і змінимо b_{l_i} на $b_{l_i} \oplus 1$. Тепер, якщо ми направимо ребро з l_i до $r_i + 1$, нам потрібно змінити b_{r_i+1} на протилежне і взяти ребро до відповіді. Якщо ми направимо ребро в іншому напрямку, ми змінюємо b_{l_i} на протилежне і не беремо запит. Тепер нам потрібно направити всі ребра так, щоб деякі вершини мали непарний, а інші парний вхідний ступінь (залежно від значення b_i). Для цього нам потрібно знайти дерево DFS і ребра, які не з дерева DFS, направити в будь-якому напрямку. Потім ми можемо жадібно направити всі ребра в дереві, починаючи з листків. Потім просто перевірте, чи всі вершини мають бажаний вхідний ступінь. Рішення працює за $O(n)$.

Задача D2. Сакурако та вкрадена легенда

Автор: **Богдан Фейса**
Підготовка задачі: **Богдан Фейса**
Розбір: **Богдан Фейса**

Блок 1: $n, m \leqslant 80$:

Ми обчислюємо всі сильно зв'язані компоненти (SCC) за $\mathcal{O}(n^3)$ для кожного з m моментів часу, використовуючи простий DFS. Після цього ми створюємо масив $ans[i][j]$, $1 \leqslant i, j \leqslant n$, який зберігає мінімальний момент часу, коли i та j знаходяться в одній SCC. Після цього ми відповідаємо на всі q запитів за $\mathcal{O}(1)$. Загальна часова складність: $\mathcal{O}(n^3 \cdot m + q)$.

Блок 2: $n, m \leqslant 10^3$:

Ми обчислюємо всі SCC за $\mathcal{O}(n + m)$ для кожного з m моментів часу, використовуючи алгоритм Косараджу. Після цього ми створюємо масив $ans[i][j]$, $1 \leqslant i, j \leqslant n$, який зберігає мінімальний момент часу, коли i та j знаходяться в одній SCC. Після цього ми відповідаємо на всі q запитів за $\mathcal{O}(1)$. Загальна часова складність: $\mathcal{O}((n + m) \cdot m + q)$.

Блок 3: $q \leqslant 100$:

Для кожного блоку ми використовуємо бінарний пошук для кожного запиту, в якому ми обчислюємо SCC $\log n$ разів за $\mathcal{O}(n + m)$ часу.

Це призводить до загальної складності часу $\mathcal{O}((n + m) \cdot q \cdot \log n)$.

Тепер ми вводимо офлайн інкрементальний алгоритми SCC, який працює за $\mathcal{O}(m \log m)$ часу.

Давайте використаємо техніку розділення та володарювання на списку ребер. Ми також можемо думати про це як про розділення та володарювання по часовій лінії. Отже, припустимо, що у нас є функція `rec(a, b, edges)`, де a — початок інтервалу часу, b — кінець (обидва включно), а `edges` — список ребер разом із часами, коли вони були додані. Ми починаємо процес з `rec(0, m-1, everything)`.

Кожного разу, коли ми дивимося на середину інтервалу часу, ми обчислюємо $s = \left\lfloor \frac{a+b}{2} \right\rfloor$. Ми використовуємо стандартний офлайн алгоритм для обчислення SCC в графі, що складається лише з ребер з `edges`, які були додані в момент до s. Ми хочемо розділити рекурсивно, не обчислюючи щось лишній раз.

Виявляється, що в моменти після s всі SCC, які ми щойно обчислили, залишаться з'єднаними. Отже, ми можемо об'єднати цілі компоненти в окремі вершини і лише передати вправо ребра, що з'єднують різні компоненти. Ребра, що з'єднують різні компоненти, були непотрібними в минулому, тому ми передаємо решту вліво. Таким чином, кожне ребро з'являється один раз на кожному рівні рекурсії. Є $\mathcal{O}(\log m)$ рівнів, і на кожному рівні загальна складність є лінійною. Отже, ми отримуємо складність $\mathcal{O}(m \log m)$ (припускаючи обережну реалізацію без додаткових логарифмів).

Блок 4: Повне рішення, без тестів з $t = 1$:

Спочатку ми створюємо додаткові $2 \cdot q$ ребра і додаємо їх до графа. Це представляє запити. Потім ми використовуємо офлайн інкрементальний алгоритм SCC, щоб отримати інформацію про кожне ребро і чи його кінцеві точки знаходяться в одній SCC в будь-який момент часу. Після цього ми витягуємо відповіді з q додаткових ребер, які ми додали.

Часова складність: $\mathcal{O}((m + q) \log(m + q))$.

Блок 5: Повне онлайн рішення:

Ми даємо вагу кожному ребру - час, коли його кінцеві точки знаходяться в одній SCC. (Вартість: $\mathcal{O}(m \log m)$). Після цього ми створюємо неорієнтовану зважену копію графа. Потім ми будуємо мінімальне остовне дерево (MST). (часова складність: $\mathcal{O}(m \log m)$)

Нарешті, ми використовуємо бінарні підйоми (бінарні стрибки) + LCA (найнижчий спільний предок), щоб знайти максимальну вагу ребра на шляху від a до b в MST. (часова складність: $\mathcal{O}(n \log n)$)

Загальна часова складність: $\mathcal{O}(m \log m + m \log m + n \log n)$