A. Prefix and Suffix Mex

Зауважимо, що $b_i \geq b_{i-1}$ та $c_i \geq c_{i+1}$, оскільки при додаванні нових значень в множину їх mex не може зменшуватись.

Якщо $b_i \neq b_{i-1}$, то повинно виконуватись $a_i = b_{i-1}$, адже жодне число зліва від i не дорівнює b_{i-1}, а разом з a_i уже хоча б одне число має бути рівним b_{i-1}. Аналогічно, $a_i = c_{i+1}$, якщо $c_i \neq c_{i+1}$.

Таким чином, ми маємо вже деякі значення зафіксовані. Подивимось, що ще необхідно, щоб виконувалась умова. Якщо $b_i \neq b_{i-1}$ також потрібно, щоб усі значення $b_{i-1}+1, b_{i-1}+2, \ldots, b_i-1$ були на префіксі $[1, i-1]$ хоча б раз. Додамо обмеження для таких значень $v \in [b_{i-1}+1, b_i-1]$, що вони повинні бути хоча б раз на проміжку $[1, l_v]$, тут $l_v = i-1$. Аналогічно зробимо обмеження, що значення v повинне бути хоча б раз на суфіксі $[r_v, n]$.

Викинемо ті обмеження, які вже виконані уже зафіксованими значеннями. Далі зауважимо, якщо для значень $u < v$ існують обмеження на префікс, то $l_u \leq l_v$. Та якщо на суфікс, то $r_u \geq r_v$. Цю властивість можна отримати напряму з того, як ми генеруємо обмеження та того, що масиви b та c неспадний і незростаючий відповідно.

Тобто, якщо $u < v$, то і префікс і суфікс на який потрібно поставити v є більшим за відповідний префікс чи суфікс для u. Це означає, що ми можемо жадібно ставити значення від найменших.

Будемо розглядати значення від 1 до n. Якщо для цього значення немає обмежень, то можна його пропустити. Якщо для значення v існують обмеження $[1, l_v]$, $[r_v, n]$, $r_l \leq l_v$ та на проміжку $[r_v, l_v]$ ще є незаповнені значення, то поставимо туди значення v. Інакше, якщо існує обмеження $[1, l_v]$, то поставимо значення v на найлівіше вільне місце, та на найправіше вільне місце, якщо існує обмеження $[r_v, n]$.

Для зберігання вільних позицій використаємо структуру даних set. Якщо в кінці залишились ще вільні позиції, то поставимо там будь-які великі значення.

Складність розв'язку $O(n \log n)$.

B. Graph on a Plane

Помітимо, що в повному графі, де є усі можливі ребра, умова про відповідність найкоротших шляхів у графі з евклідовою відстанню виконується. Проте ми можемо видалити ребро між вершинами i та j, якщо існує вершина k, яка лежить на відрізку між точками i та j. Видалення таких ребер не змінить найкоротші шляхи у графі. Можна довести, що будь-які інші ребра видаляти не можна, оскільки в такому випадку найкоротший шлях між відповідними вершинами не буде рівним їхній евклідовій відстані.

Для того аби порахувати відповідь швидко, ми для кожної вершини дивимось скільки є унікальних напрямків до інших вершин. Напрямки можна зберігати як (x, y), де x та y взаємнопрості, маючи такі пари нам треба порахувати швидко кількість унікальних.

Складність $O(n^2 \cdot \log n)$.

C. Divisible Array

Оскільки числа в масиві натуральні, то щоб b_i ділилось на $(b_{i+1} + b_{i+2})$, необхідно аби ця сума була меншою або рівною за b_i. Формально має виконуватись нерівність $b_i \geq (b_{i+1} + b_{i+2}) \geq (b_{i+1} + 1)$, а отже $b_i > b_{i+1}$. Це означає, що якщо масив впорядкований не за спаданням, то умова на подільність не виконуються.

Далі, після впорядкування, необхідно перевірити умову подільності для кожного індексу i від 1 до $n-2$. Якщо умова виконується для всіх індексів, відповідь — `Yes`, інакше — `No`.

Складність $O(n \cdot \log n)$.

D. Minimum Queries

Якщо ми зробимо перший запит на підвідрізку не зі всіх елементів, то цей запит не дасть нам достатньо інформації, оскільки ми дізнаємось лише позицію мінімуму на відрізку, але не можемо

нічого сказати про значення цього мінімуму. Отже, перший запит доцільно робити на всьому масиві, щоб одразу визначити позицію числа 1.

Якщо ми знайдемо позицію числа 1, але це число не є крайнім елементом масиву, то ми не можемо отримати достатньо інформації для визначення розташувань решти чисел. Наприклад, у масивах 2, 3, 1, 4, 5 та 4, 5, 1, 2, 3 ми можемо дізнатися лише позицію числа 1, але не зможемо визначити позиції інших чисел.

Для того аби дізнатись позицію числа $x > 1$, необхідно щоб число 1 було на крайній позиції (1 або n). Це дозволяє нам робити запити які не включають 1. У такий спосіб ми переходимо до меншої задачі — пошуку числа x у перестановці чисел від 2 до n.

Складність $O(n)$.

E. Different Adjacent

Якщо в масиві a усі сусідні елементи різні, то це масив вже задовольняє умови задачі. У інакшому випадку, якщо в масиві a є сусідні елементи, які рівні, то значення $max_{1 \le i \le n}(b_i - a_i)$ буде хоча б 1.

Будемо будувати масив b поступово:

- $b_1 = a_1$,
- для наступних елементів, якщо $a_i \neq b_{i-1}$, то ми можемо вибрати $b_i = a_i$,
- інакше оберемо $b_i = a_i + 1$, щоб зробити сусідні елементи різними.

Таким чином ми можемо побудувати масив b, який задовольняє всі умови задачі.

Складність $O(n)$.

F. Minimize the Maximum Distance

Використаємо двійковий пошук, щоб знайти відповідь. Маючи обмеження на максимальну відстань d, ми можемо що наша площина розіб'ється на такі прямокутники, в кожному з яких будь-які дві сусідні точки, по горизонталі або вертикалі, мають відстань не більшу за обмеження.

Розглянемо один прямокутник і визначимо чи можна його точки розбити на потрібні нам групи по 2 та по 3. У випадку коли в прямокутнику лише 1 точка, то ми взагалі не можемо ніяк розбити на групи. Якщо кількість точок в прямокутнику парна, то ми можемо розбити на групи по 2 і в ці групи будемо ставити пари сусідніх точок.

Останнім випадком є ситуація коли кількість точок є непарною, тоді можна довести що якщо існує розбиття на групи, то існує й розбиття на такі групи що є лише одна група із 3-х точок. Ці три точки або три підряд в одному ряді, або три підряд в одному стовпці, або усі ці 3 точки знаходяться в двох сусідніх рядках та двох сусідніх стовпцях (як у малюнку для другого прикладу сині точки).

Якщо розфарбувати такий прямокутник у шаховому порядку, припустимо що кількість чорних більша за кількість білих точок. Група з 3-х точок повинна містити дві точки чорного кольору. Єдиним випадком коли це справді важливо, коли в прямокутнику або один рядок, або один стовпець. У решті випадків якщо можна вибрати хоча б якусь групу з 3-х точок, що задовольняють умову на максимальну відстань, то й можна вибрати групу з 3-х точок, що буде задовольняти одразу умову на відстань та умову на те що дві точки чорного кольору.

Враховуючи вищесказане, нам цікаво лише прямокутники із непарною кількістю елементів. Якщо у нас утвориться прямокутник 1×1, то ми знаємо що така відстань є поганою. Після цього нам цікаво чи немає жодних проблем у прямокутниках $1 \times k$ або $k \times 1$. Наступним кроком ми можемо видалити ті прямокутники в яких можна вибрати групу з 3-х горизонтально або вертикально послідовних точок.

Серед тих що залишились потрібно перевірити чи є в кожному прямокутнику r_i та c_j такі що $\sqrt{r_i^2 + c_j^2} \le d$. Оскільки прямокутників може бути багато, знайдемо окремо $\max \min r_i$ по рядках та $\max \min c_j$ по стовпцях. Тоді достатньо перевірити чи $\sqrt{(\max \min r_i)^2 + (\max \min c_j)^2} \le d$

Складність $O((n+m)\cdot \log)$.

G. Maximize the Minimum Distance

Будемо розглядати точки в задачі як клітинки прямокутника зі сторонами n та m, а координати це індекси клітинки в такій матриці.

Розглянемо конструкцію у випадку коли n та m одночасно парні. У такій ситуації ми можемо розрізати прямокутник на 4 менші однакові прямокутники, розрізами посередині рядків та стовпців. Тепер ми можемо співставити лівий верхній прямокутник із нижнім правим, та два інші. Тобто будемо робити по 2 клітинки кожного кольору. Детальніше співставлення можна подивитись на рисунку 1 нижче.

1	2	3	7	8	9
4	5	6	10	11	12
7	8	9	1	2	3
10	11	12	4	5	6

Рис. 1: Відповідь для $n = 4$, $m = 6$

Якщо ж одне з чисел n та m є парним, а інше непарне, то розрізати на прямокутники треба дещо інакше. Наприклад, коли кількість рядків непарна, то ми можемо ліву частину центрального рядка долучити до лівого верхнього прямокутника і праву частину до правого нижнього прямокутника. Детальніше співставлення можна подивитись на рисунку 2 нижче.

1	2	3	10	11	12
4	5	6	13	14	15
7	8	9	1	2	3
10	11	12	4	5	6
13	14	15	7	8	9

Рис. 2: Відповідь для $n = 5$, $m = 6$

У випадку коли обидва числа непарні, то ми зробимо подібний процес зі стовпцем. Проте станеться така ситуація, що центральну клітинку покривають і лівий верхній, і нижній правий прямокутники. То пофарбуємо ліву верхню клітинку, праву нижню клітинку і центральну в один колір. Детальніше співставлення можна подивитись на рисунку 3 нижче.

1	2	3	4	12	13	14
5	6	7	8	15	16	17
9	10	11	1	2	3	4
12	13	14	5	6	7	8
15	16	17	9	10	11	1

Рис. 3: Відповідь для $n = 5$, $m = 7$

В усіх випадках така конструкція дає найкращу можливу відповідь, оскільки для кожної з центральних клітинок неможливо отримати відстань кращу за $\lfloor \frac{n}{2} \rfloor^2 + \lfloor \frac{m}{2} \rfloor^2$.

Складність $O(n \cdot m)$.

H. Laminaria

З'єднувати можливо тільки вершини однакового кольору, інакше неможливо зробити першу операцію.

Розглянемо таку інтерпретацію процесу з умови: назвемо спершу великою вершиною – вершину, яка з'єднує обидва дерева. Далі за кожну операцію будемо об'єднувати велику вершину з певною вершиною u, якщо в обох деревах існує ребро між великою вершиною і вершиною u. Усі ребра, які виходили зі старих вершин, тепер будуть виходити з нової великої вершини.

Зробимо схожий процес: поки у двох деревах є однакове ребро, об'єднаємо в обох деревах ці вершини в одну вершину. Якщо такий процес не об'єднав усі вершини в кожному з дерев в одну, то відповідь 0, адже в одну велику вершину теж не вийде об'єднати. Такий процес не залежить від початкової великої вершини, тож його можна зробити один раз.

Щоб реалізувати цей процес будемо підтримувати усі інцидентні ребра для вершини в списку та з'єднувати ребра від вершини з меншим степенем до вершини з більшим степенем. Складність такого алгоритму буде $O(n \log^2 n)$, оскільки нам ще також потрібно використати set чи схожу структуру даних, щоб підтримувати, які ребра збігаються в обох деревах.

Подивимось, коли ми об'єднували вершини u та v, які ребра відповідають цьому об'єднанню в оригінальних деревах. Нехай це (u_1, v_1) в першому дереві та (u_2, v_2) в другому. Якщо $u_1 \neq u_2$, то якщо велика вершина спершу об'єднала вершини v_1 чи v_2, то ми не зможемо об'єднати також u_1 і u_2. Аналогічно, якщо $v_1 \neq v_2$. Тож якщо $u_1 = u_2$, додамо в додатковий граф напрямлені ребра $v_1 \to u_1$, $v_2 \to u_1$. Це означає, що якщо ми об'єднали в велику вершину вершини з v, то зможемо об'єднати також вершини з u. Аналогічно додамо напрямлені ребра, якщо $v_1 = v_2$. Тоді відповідь – це кількість вершин, з яких досяжні всі інші вершини в додатковому графі. Таке значення можна знайти, якщо зробити топологічне сортування, або також існують простіші методи, які враховують специфічну структуру додаткового графа.

I. Minimum Divisible Sequence

Для чисел $k > \frac{n}{2}$ є лише одне можливе значення для a_i, яке є множником k, це число k. Це означає, що в масиві повинні бути всі числа від $\lfloor \frac{n}{2} \rfloor + 1$ до n. Таким чином масив матиме щонайменше $\lceil \frac{n}{2} \rceil$ елементів.

Для кожного числа $k \leq \frac{n}{2}$ можна знайти число в інтервалі $[\lfloor \frac{n}{2} \rfloor + 1, n]$, яке ділиться на k. Це випливає з того що розмір проміжку більший або рівний за k. Враховуючи вищесказане, масив з елементів від $\lfloor \frac{n}{2} \rfloor + 1$ до n є мінімальним можливим масивом, що задовольняє усі умови задачі.

Складність $O(n)$.

J. Permutations of Two Arrays

Без обмеження загальності вважатимемо, що $n <= m$. Впорядкуємо масив a у порядку зростання, а масив b у порядку спадання. Оскільки ми хочемо максимізувати відстань, то з масиву b вигідно брати лише елементи на префіксі та суфіксі. Переберемо len —кількість елементів з префіксу b ми беремо, та паруємо їх з суфіксом масиву a. Відповідно префікс масиву a довжини $n - len$ паруємо з суфіксом масиву b. Тоді відповідь буде рівною

$$\max_{len \in [0,n]} \left(\sum_{i=n-len}^{n-1} a_i - \sum_{i=0}^{len} b_i \right) + \left(\sum_{i=m-len}^{n-1} b_i - \sum_{i=0}^{len} a_i \right)$$

.

Таку формулу можна рахувати за допомогою часткових сум, або перераховувати доданки в процесі зміни len.

Складність $O(n)$.

K. Values in a Rooted Tree

Будемо рахувати відповідь в процесі обходу дерева в глибину. При вході у вершину потрібно додати її значення до множини, а при виході видалити. Тепер після входу потрібно знайти відповідь. Подивимося як зміниться відповідь, якщо ми перейшли від батька до сина. Нехай ми прийшли до вершини v. Відповідь може або не змінитися, або збільшитися, якщо найдовший відрізок буде містити значення a_v.

Для того аби знайти найдовший відрізок зі значенням a_v, і оновити ним відповідь, можна використати сет. У цьому сеті ми можемо зберігати відрізки $[l, r]$, такі що на шляху від кореня до цієї вершини були усі значення від l до r включно. Тепер при вході у вершину v цей сет може змінитись в один із чотирьох способів:

- Ніяк не зміниться, якщо число a_v уже зустрічалось на шляху до кореня.
- Утвориться новий відрізок довжини 1 — $[a_v, a_v]$.
- Довжина якогось відрізка збільшиться на 1. Такий відрізок мав мати або $r = a_v - 1$, або $l = a_v + 1$.
- Два відрізки об'єднаються в один, відрізки $[l, a_v - 1]$ та $[a_v + 1, r]$ об'єднаються в один $[l, r]$.

Коли ми будемо виходити з вершини v, нам треба скасувати усі зміни які ми робили в цій вершині. Ми можемо запам'ятати які відрізки ми видалили з сету при вході і які додали. При виході видалимо ті що додали і додамо ті що видалили.

Складність $O(n \cdot \log n)$.

L. L-trominos

Використовуючи замощування на рисунку 4 ми можемо досягти $2 \cdot n \,\%\, 3$ вільних клітинок.

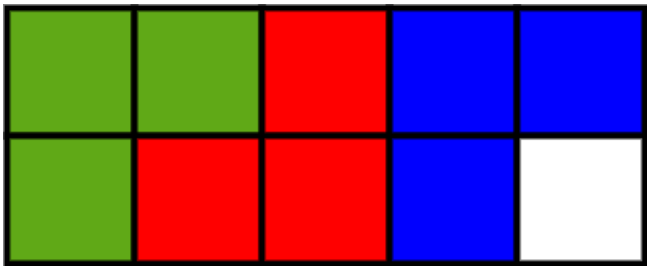

Рис. 4: Приклад замощення

Складність $O(1)$.