Розбір задачі «Мінімум»

Підзадача 1

Нехай $k = \min(a_1, b_1)$, тоді $x = [k, k]$.

Підзадача 2

Оскільки m мале, то можна перебрати можливі значення. Якщо для певного числа немає обмеження, то воно може бути довільним.

Підзадача 3

Якщо $\min(x_{a_i}, x_{b_i}) = c$, тоді ми знаємо, що $x_{a_i} \geqslant c$ і $x_{b_i} \geqslant c$. Тоді $x_i = max(c_j)$ для усіх j таких, що або $a_j = i$, або $b_j = i$.

Це можна зробити за допомогою двох циклів. Тому складність буде $O(n^2)$.

Підзадача 4

Оскільки $c_i \leqslant 10$, то існує рішення, де кожне число від 1 до 10. Ми можемо перебрати усі можливі варіанти, яких 10^n. Та перевірити чи всі умови виконуються. Зверніть увагу, що можуть бути кілька обмежень для однієї і тієї самої пари. Тому у нас буде не більше n^2 унікальних обмежень. Очевидно, що можна перевіряти лише їх. Тому складність перевірки на правильність буде $O(n^2)$. Сумарна складність буде $O(10^n \cdot n^2)$.

Підзадача 5

Така сама ідея, як і в підзадачі 3, але можна покращити складність до $O(n)$, запустивши один цикл з m обмеженнями та одночасно збільшуючи x_i, якщо є потреба.

Розбір задачі «Лазери»

Підзадача 1

У цій підзадачі $R = 1$, $X = 1$,тобто у нас є тільки один рядок і одна стіна. Помітимо, що якщо довжина стіни дорівнює w, то відрізок $(L - w, w)$ завжди буде заблокований. Тому відповідь буде $max(0, 2w - L)$.

Підзадача 2

Щоб вирішити задачу без обмеження $R = 1$, зауважте, що лише найдовша стіна має значення, оскільки кожна інша стіна може "ховатися"за довшою стіною. Отже, ви можете визначити максимальну довжину стіни, і потім повернутися до підзадачі 1.

Підзадача 3

Щоб перевірити, чи стовпець вільний принаймні в одному розташуванні, ми перебираємо кожен рядок і зсуваємо стіни до лівого краю рядка, поки це можливо зробити без блокування поточного стовпця. Всі інші стіни, які не помістилися ліворуч, ми зсуваємо праворуч. Якщо нам вистачило місця для стін праворуч, то в цьому рядку ми можемо зробити поточний стовпець вільним. Щоб перевірити, чи стовпець може бути вільним в якомусь розташуванні, перевіримо, чи він може бути вільним в обох рядках.

Замість того, щоб кожен раз виконувати зсування стін ліворуч та праворуч, ми можемо створити окремий масив для кожного ряду, що містить суму довжин стін на префіксі. За допомогою бінарного пошуку, можна легко і достатньо швидко знаходити кількість стін, які помістяться ліворуч. Отже, загальна складність рішення становить $\mathcal{O}(LRlogX)$.

Підзадача 4

Рішення підзадачі 3, можна легко розширити, щоб вирішити підзадачу 4. Просто перевіряємо для кожного стовпця всі рядки, а не лише два.

Підзадача 5

У цій підзадачі $1 \leqslant L \leqslant 10^6$. Ми можемо розширити рішення підзадачі 4, щоб вирішити цю. Перш за все ми розглянемо розташування, коли всі стіни зсунуті вліво, тобто коли стовпчик, який ми розглядаємо, є L-им стовпцем. Тепер будемо рухати стовпчик, який ми розглядаємо ліворуч. Коли стовпець пересунувся ліворуч на 1, ми повинні усі стіни, які зараз закінчуються на цьому ж стовпці пересунути вправо, при цьому запам'ятовуючи максимальну сумарну довжину стін, пересунутих вправо серед усіх рядків. Це рішення працює в часі $\mathcal{O}(L + \sum X)$.

Підзадача 6

Зауважимо, що для кожного ряду ми можемо знайти множину стовпців, що можуть бути вільними в цьому стовпці. Ця множина — об'єднання множин для кожного $1 \leqslant i \leqslant X$ $(l_1 + \ldots + l_i, L - l_{i+1} - \ldots - l_X) - (L - l_i - \ldots - l_X, l_1 + \ldots + l_i)$.

Щоб використати це рішення для декількох рядків, ми просто розглянемо обернення цих множин у кожному рядку (тобто набір стовпців, які заблоковані, а не вільні), і потім зробимо об'єднання цих множин по всіх рядках. Це рішення працює в часі $mathcalO(\sum Xlog \sum X)$.

Розбір задачі «Перестановка»

Підзадача 1

В цій підзадачі $1 \leqslant n, m \leqslant 9$. Одним із підходів до цієї задачі було перебрати усі можливі перестановки, перевірити які з них мають мінімальне кістякове дерево, що складається з ребер b, та обрати з них мінімальну лексикографічно.

Підзадача 2

Підзадача 2 може бути вирішеною неоптимальною версією фінального рішення. Щоб знайти лексикографічно мінімальну перестановку W треба спочатку мінімізувати W_1, потім мінімізувати W_2, і так далі.

Ми перебираємо ребра у порядку зростання номера та робимо наступне:

- Якщо ребро вже має вагу, ігноруємо його.
- Інакше, якщо ребро належить b, ставимо йому мінімальну вагу, яку ми ще не використовували.
- Інакше, якщо ребро (a, b) не належить b, ми розглядаємо шлях (a, b) у дереві з ребер b. Оскільки усі ребра на цьому шляху повинні мати меншу вагу ніж ребро (a, b), ми можемо розглядати тільки ребра, що ще не мають вагу. Кожному з цих ребер назначимо мінімальний не зайнятий номер, у порядку зростання номера ребра. Потім ребру (a, b) видамо мінімальний не зайнятий номер.

Обробка кожного ребра займає $O(n)$ операцій, наївним алгоритмом. Асимптотика такого рішення $O(n \times m)$.

Підзадача 3

Рішення для другої підзадачі так само буде працювати для третьої, оскільки на шляху (a, b) буде не більше двох ребер. Тоді кожне ребро буде опрацьованим за $O(1)$ операцій, а загальна асимптотика такого рішення $O(m)$, і реалізовувати таке рішення легше ніж попереднє.

Підзадача 4

В даній підзадачі ребра b утворюють лінію. Це спрощує реалізацію другої підзадачі, оскільки ми можемо підтримувати в сеті усі позиції на яких ще не видалене ребро. І одразу як ми назначаємо ребру вагу, видаляємо його із сету. Така реалізація буде мати асимптотику $O(m\ log\ m)$.

Підзадача 5

В даній підзадачі граф утворює цикл. Єдиному ребру що не належить b можна дати вагу n, а усім іншим ребрам вагу від 1 до $n - 1$, у порядку зростання номеру ребра.

Підзадача 6

В даній підзадачі $n = m$. Нехай єдине ребро, що не належить b, це ребро (a, b). У зростаючому порядку ваги, ми помічаємо усі ребра, що мають номер менше ніж номер ребра (a, b). Потім помічаємо усі ребра що лежать на шляху (a, b), у тому ж порядку. Далі помічаємо ребро (a, b), і потім усі інші ребра.

Підзадача 7

Фінальне рішення включає оптимізоване рішення другої підзадачі. Ми можемо використовувати систему незалежних множин, щоб знаходити лише непомічені ребра на шляху (a, b).

Для цього, коли ми помічаємо ребро, ми об'єднуємо його кінці в одну вершину системи незалежних множин, де предком підвішеної вершини ми обираємо вершину що ближче до кореня дерева. За корінь при цьому на початку можна взяти будь-яку вершину. Коли нам треба помітити ребра на шляху, ми можемо по черзі брати вершину з більшою глибиною, та помічати ребро до її предка, поки обидві вершини не зустрінуться в їх найменшому спільному предку. Коли ми знайдемо усі ребра на шляху (a, b) їм можна видати вагу у порядку зростання їх номера. Загальна асимптотика такого рішення $O(m\ log\ m)$.

Розбір задачі «Тасування»

Підзадача 1

У цій підзадачі $b = 2$ та $k = 3$.

Існує $C_6^3 \div 2! = 10$ можливих унікальних запитів, при цьому маємо обмеження $q = 100$. Також розуміємо, що існує $n! = 720$ різних варіантів x. Можна згенерувати усі 10 запитів. Потім проітеруємося по всім можливим x і промоделюємо 10 запитів, порівнюючи результати моделювання і результати повернені програмою. Коли всі результати співпадають, тоді x визначено.

Підзадача 2 У цій підзадачі $b = 3$ і $k = 2$.

Існує $C_6^2 \times C_4^2 \div 3! = 15$ можливих унікальних запитів. x знову можна визначити методом перебору усіх значень, описаним в Підзадачі 1.

Підзадача 3 У цій підзадачі $q = 12$ і порядок b множин невипадковий (порядок не змінюється).

Для кожного запиту, m-а група з k чисел у запиті якимось чином співставляється з m-ою групою з k чисел у результаті запиту. Якщо число i зустрічається у m_1-ій, m_2-ій, ..., m_q-ій множинах серед q запитів, тоді число x_i також зустрічається у тих самих m_1-ій, m_2-ій, ..., m_q-ій множинах серед q результатів запитів.

Ціль цієї підзадачі — знайти якусь послідовність запитів таку, що кожне число i від 1 до n має унікальну послідовність $m_1, m_2, ..., m_q$ множин у яких воно з'являється. x_i визначається як число із результатів запитів, що має таку саму послідовність.

Враховуючи обмеження, ми можемо звести задачу до такої:

Згенерувати N унікальних чисел (назвемо їх ідентифікаторами) у системі числення з основою b таких, що кожна цифра від 0 до $b - 1$ зустрічається на кожній позиції рівно k разів.

З кожним запитом ми використовуємо одну нову цифру у всіх ідентифікаторах. За оптимального генерування, максимальна довжина ідентифікатору (і відповідно оптимальна кількість запитів) буде не більшою за $\lceil \log_b(n) \rceil$

Підзадача 4

У цій підзадачі $k = 2$ і $q = 4$. Зведемо підзадачу до задачі на графах. Нехай кожне число у запитах це вершина графа. Знаходження чисел в одній множині розміру $k = 2$ будемо розглядати як неорієнтоване ребро між ними. Помітимо, що якщо у графі запиту присутнє ребро між i та j, то у графі результату запиту буде присутнє ребро (x_i, x_j). Декілька запитів будемо об'єднувати у один граф із різними кольорами ребер.

В результаті ми отримуємо два ізоморфні графи з неорієнтованими, зафарбованими ребрами: один граф ми отримали із запитів, і один із результатів на ці запити.

Щоб визначити x_i необхідно, щоб вершина i у графі запитів мала таку ж саму зв'язність як і вершина x_i у графі результатів. Щоб визначити x, треба, щоб усі вершини однозначно визначались за їх зв'язністю, тобто щоб граф не мав автоморфізму.

З маленькою кількістю запитів, інтуїтивним кроком може бути створення однієї компоненти зв'язності у найменшу кількість запитів. Циклічний граф можна створити за 2 запити:

- $[1, 2], [3, 4], [5, 6], ..., [n - 3, n - 2], [n - 1, n]$ для першого запиту, ребра фарбуємо у червоний
- $[2, 3], [4, 5], [6, 7], ..., [n - 2, n - 1], [n, 1]$ для другого запиту, фарбуємо у синій.

В результаті маємо циклічний граф з червоними та синіми ребрами, але цей граф ще досі має автоморфізм. Щоб зруйнувати симетрію, два додаткових запити можна використати для однозначної ідентифікації «кореня». Існує декілька можливих методів це зробити, і один із них — визначити вершину 1 як корінь:

- Змішуємо перші дві групи першого запиту: $[1, 3], [2, 4], [5, 6], ..., [n - 3, n - 2], [n - 1, n]$ — третій запит, ребра фарбуємо у зелений.
- Змішуємо перші дві групи другого запиту $[2, 4], [3, 5], [6, 7], ..., [n - 2, n - 1], [n, 1]$ — четвертий запит, фарбуємо у жовтий.

Ці третій та четвертий запити згенерують ті самі ребра що і в першому та другому запитах відповідно, окрім перших двох груп. Після усунення повторів (зелений-червоний) та (синій-жовтий)

ребер, помітимо, що вершина 1 у графі запитів і відповідно x_1 у графі результатів будуть з'єднані з зеленими ребрами, проте не будуть з'єднані з жовтими. Граф більше не має ізоморфізмів. Залишок x може бути визначений ітеруванням по червоним та синім ребрам у графі результатів, починаючи з x_1.

Підзадача 5 У цій підзадачі $b = 2$ і $q = 12$.

Використовуючи всього три запити можна визначити одне із значень x, наприклад x_1. Для наступних запитів завжди кладіть «корінь» (x_1) у першу множину. Тоді який би результат не отримався, множина, що містить x_1 це перша множина, і всі її елементи якось співставляються з першою множиною запиту. Випадковість порядку множин скасована, а тому ми звели задачу до Підзадачі 3.

Ця підзадача може бути розв'язана у $3 + \lceil \log_b(n) \rceil$ запити. Оптимізації (повторне використовування даних запитів) можуть зменшити це значення до 12 у найгіршому випадку.

Підзадача 6 Розглянемо більш загальну версію зведення задачі до графу. Для довільного k ми будемо використовувати техніку описану у Підзадачі 4, але при цьому будемо «об'єднувати вершини». Наприклад, множину з k чисел будемо розглядати як «корінь» з номером 1 і колективну супервершину з номерами $[2, 3, ..., k]$. Так як тепер кожна множина містить лише дві «ефективні» вершини, то ми можемо перейти до використання техніки Підзадачі 4, щоб визначити x_1 й інші вершини серед b «коренів».

Примітка: техніка описана в Підзадачі 4 тепер використовує лише 3 запити замість 4-ьох, тому що нерівномірні розміри «кореня» і супервершини одразу руйнують симетрію у графі.

Як тільки ми дізнаємося значення b «коренів» у x, ми зможемо використати техніку описану в Підзадачі 5. Ми покладемо перший «корінь» у першу множину, другий «корінь» у другу, і так далі до b-ої множини. Коли ми отримаємо результат запиту порядок множин може бути визначений за рахунок вже відомих нам b коренів. Таким чином, ми звели задачу до Підзадачі 3.

Ця підзадача може бути вирішена у $3 + \lceil \log_b(n) \rceil$ запити. Подальші оптимізації зменшать це значення до 9 у найгіршому випадку.