Задача A. Рабочая атмосфера

Ответ на задачу — N и N^2. Построим примеры на каждый из них:

1	2	3	4
4	1	2	3
3	4	1	2
2	3	4	1

Вначале заполняем одну диагональ, там будут только 1, затем сдвигаем её на один вправо, заполняем получившиеся клетки, там будут только 2, и так далее. В итоге результатом будет N.

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

Вначале заполняем первую строку слева направо — получатся числа от 1 до N, затем вторую строку справа налево — получатся числа от $N+1$ до $2 \cdot N$, и так далее. Результатом будет $N \cdot N$.

Задача B. Две прогрессии

Сначала разберём случай, когда одна из прогрессий состоит из двух или более одинаковых чисел. Найдём любое число, которое встречается больше одного раза, и попытаемся взять в одну прогрессию все такие числа или все кроме одного. После чего проверим, что все остальные числа образуют прогрессию любого типа. Проверять несколько куч одинаковых чисел не имеет смысла: если первая попавшаяся куча не входит в одну прогрессию, то мы не найдёт ответа в другой куче.

Осталось рассмотреть случай, когда обе прогрессии состоят из разных чисел. Для этого нам понадобится ключевое утверждение: пусть $a \leqslant b \leqslant c$ — три подряд идущих числа из арифметической прогрессии. Тогда они не могут все вместе входить в геометрическую прогрессию. Докажем от противного: пусть все они входят в геометрическую прогрессию. Тогда выполняется $b = a \cdot q^x$ и $c = a \cdot q^y$, где $q > 1$ — шаг геометрической прогрессии, а x и y — некоторые натуральные числа. Для арифметической прогрессии выполняется $b - a = c - b$, или $a = 2b - c$. Отсюда $a = a \cdot (2q^x - q^y)$. Но правая часть равенства точно делится на $a \cdot q$ (или на a умноженное на числитель q, если q — рациональное число), а левая — нет. Получили противоречие.

Используя это утверждение, можно сконструировать следующий подход к задаче: пусть мы нашли два различных числа, которые идут подряд в арифметической прогрессии. Тогда мы можем расширить прогрессию влево и вправо до тех пор, пока нужные числа встречаются в данном наборе. Получим последовательность $b_1, b_2, \ldots, b_k$. Но эта последовательность не обязательно является ответом на задачу: мы, возможно, захватили несколько чисел с начала и с конца, которые должны попасть в геометрическую прогрессию. Но из только что доказанного утверждения следует, что с каждого конца имеет смысл выкинуть не более двух чисел. Если бы мы выкинули три числа с начала или с конца, то они точно не могли бы принадлежать одной геометрической прогрессии. Переберём все 9 способов выкинуть числа и проверим их.

Найти пару чисел из арифметической прогрессии можно несколькими способами.

Способ 1: Отсортируем числа $a_1, \ldots, a_n$ и переберём четыре пары (a_1, a_2), (a_1, a_3), (a_2, a_3), (a_n, a_{n-1}). Если среди этих пар мы не нашли ответ, то значит среди a_1, a_2, a_3 есть хотя бы два числа из геометрической прогрессии, и среди a_{n-1}, a_n — хотя бы одно. Значит, мы можем перебрать начало геометрической прогрессии среди пар (a_1, a_2), (a_1, a_3) или (a_2, a_3) и продолжить прогрессию до a_{n-1} или a_n, а из оставшихся чисел попытаться сложить арифметическую прогрессию. Если мы всё ещё ничего не нашли, то мы исчерпали все варианты, и ответ — -1.

Способ 2: Воспользуемся ещё одним утверждением: если геометрическая прогрессия состоит из разных чисел, то в ней не более 30 чисел. Это потому, что все числа до 10^9, и шаг прогрессии обязательно умножает или делит на простое число, не меньшее 2. Для $N \leqslant 61$ посчитаем как угодно, например, переберём первый элемент, второй элемент и длину арифметической прогрессии, и проверим. Это можно сделать за $O(N^4)$. Если $N \geqslant 62$, отсортируем числа и посмотрим на все пары подряд

идущих. Среди них $N-1$ пар, и не более 60 пар содержат число из геометрической прогрессии. То есть, хотя бы $N-61$ из $N-1$ пар полностью лежат в арифметической прогрессии. Если возьмём случайную пару, то с вероятностью $\frac{60}{N}$ не угадаем. Будем повторять брать пару, пока вероятность не угадать больше какого-нибудь маленького числа, например, 10^{-6}, проверяя на каждом шаге.

Задача C. Сочинение

Рассмотрим ориентированный граф, построенный на встречающихся в вводе словах, где рёбра — заданные правила. Отметим, что ответ достигается, когда каждое слово встречается не более одного раза — если слово встречается хотя бы два раза, то можно оставить только его последнее вхождение, и длина сочинения уменьшится. А если мы вынуждены взять слово дважды, значит, в графе есть цикл, и ответа не существует.

Тогда задача о сочинении сводится к следующему: придумать некоторую последовательность вершин графа, в которой для всех правил (a, b) вершина a стоит раньше, чем b. По сути, требуется топологическая сортировка графа, которая строится при помощи обхода в глубину. Отметим, что топологическая сортировка должна быть применена не ко всему графу, а только к ключевым словам и зависимых от них — именно эти слова должны попасть в ответ.

Но нам нужна не просто топологическая сортировка, а именно лексикографически минимальная. Такую сортировку можно построить жадно при помощи кучи или любого упорядоченного множества. Будем поддерживать в ней все слова, в которые не ведут рёбра. На каждом шаге нужно взять минимальное слово, удалить исходящие из него рёбра и, возможно, добавить в кучу новые слова, у которых было удалено последнее входящее ребро. Это решение требует хранения строк сжато (при помощи хеша или номера).

Задача D. Гигант-вандал

Самое трудное в этой задаче — организовать ввод. После этого нужно посмотреть на разницы между последовательными выбитыми окнами, а также на самый нижний этаж с выбитым окном. Очевидно, что ответ будет равняться максимуму между этими величинами (так он может пролезть по очереди во все выбитые окна снизу вверх по порядку), если он был бы меньше, то гигант не смог бы перелезть в последовательную пару выбитых окон с максимальной посчитанной величиной.

Задача E. Игра в кальмара

Пусть есть два индекса a и b такие, что среди записей нет ни пары a b, ни пары b a. Покажем, что в этом случае ответ «`NO`». Построим две перестановки: в первой перестановке значения с индексами a и b равны $n-1$ и n соответственно, во второй — n и $n-1$ соответственно. Остальные значения будут какие угодно, но одинаковые в обоих перестановках. Тогда для обеих перестановках ответы будут одинаковые, и Йокай ошибётся.

Если же все пары индексов от 1 до n встречаются среди m записей, то очевидно, что всегда можно восстановить перестановку, посчитав по ответам сколько раз число было больше других.

Задача F. Мета-условие

Задача состоит из трёх частей. Основная идейная часть — это динамическое программирование, где для каждого рассматриваемого шаблона ищется ответ. Начнём с шаблона, покрывающего весь данный массив. Пусть у текущего шаблона есть левый подшаблон из L чисел, и для него мы насчитали d_l эквивалентных подшаблонов; и есть правый подшаблон из R чисел, для которого мы насчитали d_r эквивалентных подшаблонов. Поскольку левый и правый подшаблон никаким образом не связаны друг с другом, то можно произвольным образом распределить между ними $L+R$ чисел, и для каждого выбрать один подшаблон из эквивалентных. Всего получаем $d_l \cdot d_r \cdot C_{L+R}^{L}$ способов, где C_n^k — количество сочетаний из n по k. Получается, надо посчитать значение динамического программирования для левого и правого подшаблона рекурсивно, после этого ответ легко считается для текущего шаблона.

Вторая существенная часть — поиск левого и правого подшаблона. Каждый шаблон — это подотрезок исходного массива, который разбивается на подшаблоны по минимальному элементу. Получается, что каждый шаблон однозначно задаётся минимальным элементом, поэтому динамическое

программирование можно считать в простом массиве длины N. Искать минимальный элемент можно за $O(Nlog(N))$ любой структурой данных на минимум, но возможно это сделать за $O(N)$ — это стандартный алгоритм построения Декартового дерева в режиме оффлайн.

Осталось научиться считать C_n^k по модулю $10^9 + 7$. Зная формулу

$$C_n^k = \frac{n!}{k!(n-k)!}$$

можно посчитать первые N факториалов по модулю $10^9 + 7$. Но выполнять деление нужно не буквально, а тоже по модулю. Для каждого факториала $k!$ найдём обратный факториал — такое число f_k, что $k! \cdot f_k \bmod 10^9 + 7 = 1$. Поскольку $10^9 + 7$ — простое число, то из малой теоремы Ферма следует, что $f_k = k!^{10^9+5} \bmod 10^9 + 7$. Возвести в большую степень можно бинарным возведением. При этом достаточно это сделать только один раз для f_k, а для остальных обратных факториалов воспользоваться формулой $f_{k-1} = f_k \cdot k$. Итого, решение этой задачи работает за $O(N)$.

Задача G. Mount & Blade

Для начала разберёмся, в каком случае рыцарь может стать абсолютным победителем. Так как от каждого соперника можно ожидать максимальную прокачку в копьё или в щит, то *сокрушить* противника мы можем только тогда, когда и выигрываем рыцаря с параметрами $(a_i + g_i, d_i + g_i)$.

Чтобы победить всех, нам потребуется найти максимальную величину по новой атаке и защиты среди врагов. Теперь лишь остаётся понять, сколько золота нужно вложить текущему рыцарю — улучшить копьё на величину не меньше защиты, а щит больше атаки противника.

Как же можно найти максимумы? Вариант первый — перебрать остальных каждый раз, но в таком случае будет сложность $O(n^2)$, что не подходит. Вариант второй — подсчитать их на префиксе и суффиксе, получая асимптотику $O(n)$.

Альтернативное решение:

Заметим, что если i-й рыцарь — абсолютный победитель, то для любого другого j-го выполняется неравенство: $a_i + d_i + g_i > a_j + d_j + 2 \cdot g_j$. В таком случае найдём того, чья сумма параметров максимальная, и проверим только его. Как это сделать — было рассмотрено в первом решении.

Задача H. Графомания

Переберём начальную позицию графа, после чего мы знаем N и M, и нужно проверить, что далее идут $2 \cdot M$ пар рёбер с вершинами от 1 до N без петель и кратных рёбер. Для этого заведём структуру данных, позволяющую считать максимум на отрезке — это может быть дерево отрезков или sparse table.

Нам понадобятся две такие структуры, одна — для подсчёта максимального числа на отрезке и проверки, что это число не превосходит N. При добавлении чисел в неё можно вместо нуля добавлять $10^9 + 1$, потому что это число заведомо больше N.

Другая структура на максимум нам нужна для поиска кратных рёбер. Для этого для каждого ребра будем хранить предыдущую позицию, где это ребро встречалось на позиции той же чётности. Это удобнее делать, если решать задачу в два прохода: сначала — по нечётным позициям, потом — по чётным. Поместим в структуру данных для каждой позиции индекс предыдущего ребра, и если на отрезке с L по R есть число, большее или равное L, то на этом отрезке есть повтор. Не забудьте, что (u, v) и (v, u) — одно и то же ребро.

Осталось проверить петли. Это сделать можно как в первой, так и во второй структуре: вместо петли можно записать число $10^9 + 1$, и тогда это войдёт в проверку других случаев.

Задача I. Собеседование

Во-первых, нужно проверить, что сумма по строкам равна сумме по столбцам. Если это не выполняется, то ответ 0.

Если это условие выполняется, то найдется хотя бы одна прямоугольная матрица, которая удовлетворяет ограничениям. Это несложно показать индукцией по числу строк: если матрица содержит только одну строку, то ответ очевиден. Если в матрице n строк, то заполним самую верхнюю из них любым подходящим способом так, что в сумме в строке получилось соответствующее число,

а в каждой клетке в строке было не более чем значение в столбце. Удалим эту строку и вычтем из значений в столбцах числа в клетках. Получили матрицу на одну строку меньше, которую умеем заполнять по индукции.

Теперь посчитаем ответ. Очевидно, что должно выполняться $a_{ij} \leqslant r_i$ и $a_{ij} \leqslant c_j$. Для остальных чисел в строке есть аналогичное условие: если S — сумма по всей таблице, то $r_i - a_{ij} \leqslant S - c_j$. То есть, все остальные числа в строке не больше суммы сумм по их столбцам. Наконец, должно выполняться симметричное условие $c_j - a_{ij} \leqslant S - r_i$, но на самом деле это условие совпадает с предыдущим.

Если все эти условия выполняются, то, как мы уже доказали по индукции, такая матрица существует. Тогда a_{ij} может принимать значения от $\max(r_i + c_j - S, 0)$ до $\min(r_i, c_j)$.

Задача J. Букмекер

Задача состоит в аккуратной реализации того, что происходит. Заметим, что если игра начиналась с i-го символа, то не позже чем через 63 символа она закончится. Причём если один из игроков победил до этого, то никакая подстрока после этого не может являться записью игры. Но даже если продолжить выигранную игру, то и через 63 хода суммарно можно увидеть на поле сложившуюся ранее победу.

В итоге нужно проверить запись начиная с каждого символа на 63 позиции вперёд (либо до конца записи). Запись на победу и ничью можно проверить за не более чем 600 операций. Чтобы не строить игру по записи заново, можно заметить, что соседние записи отличаются только первым ходом. Можно удалить этот первый ход и сместить соответствующий столбец вниз, что делается не более чем за 63 операции. Чтобы иметь доступ к первому ходу и добавлять ходы в конец, можно завести очередь ходов.

Таким образом, вся программа работает за $O(C \cdot N)$, где C — константа, не превосходящая 700.

Задача K. Обер-форшнейдер

Задача решается с помощью динамического программирования по подотрезкам. Обозначим вершины многоугольника как $A_1, A_2, \ldots, A_n$. Пусть $dp[i][j]$ — максимальная площадь минимального треугольника в триангуляции многоугольника $A_i A_{i+1} \ldots A_j$, где $i < j$. По построению на каждой стороне многоугольника должен лежать треугольник. Переберём, какой треугольник лежит на стороне $A_i A_j$. Пусть A_k — его третья вершина. Тогда

$$dp[i][j] = \max_{i<k<j}(\min(dp[i][k], S(A_i A_k A_j), dp[k][j])),$$

где $S(A_i A_k A_j)$ — площадь треугольника $A_i A_k A_j$. Удобно считать, что $dp[i][i+1]$ равно бесконечности, чтобы автоматически рассмотреть случай когда $k = i + 1$ или $k = j - 1$.

Таким образом, нужно посчитать $O(N^2)$ состояний динамики — для каждого подотрезка по возрастанию длины. Переходы считаются за $O(N)$ каждый. Ответ будет находиться в $dp[1][N]$, суммарная сложность работы — $O(N^3)$.

Задача L. Круглый стол

Заметим, что можно считать расстояние между людьми в виде угла, а не в виде расстояния явно. Чтобы посчитать это расстояние нужно выбрать наименьший из углов между людьми по часовой стрелке и против часовой стрелки. Затем из трёх углов нужно выбрать наименьший, если таких несколько, нужно выбрать старшего.