Задача A. K-интересные подотрезки

Чтобы решить первую подгруппу, нам достаточно отвечать на запросы за $O(n^2)$. Как можно это сделать? Запросы второго типа обрабатываются несложно: давайте будем менять нужный элемент в нашем массиве a. Это будет работать за $O(1)$. Чтобы обработать запросы первого типа, давайте перебирать количество элементов, которые будут входить в сумму, затем считать эту сумму и произведение элементов справа и, наконец, проверять, что произведение этих частей действительно равно k. Итоговая асимптотика такого решения — $O(qn^2)$.

Для второй подгруппы существует множество различных подходов, давайте обсудим один из них. Воспользуемся тем, что у нас есть только запросы первого типа. А именно, предпосчитаем для каждого подотрезка все такие k, что этот подотрезок является k-интересным, а затем отсортируем их по возрастанию. Это можно сделать за $O(n^3 \log n)$, перебирая подотрезок, а затем перебирая количество элементов в правой части. После того как мы сделали такой предпосчёт, мы можем отвечать на запросы за $O(\log n)$, используя бинарный поиск в нужном массиве, чтобы проверить, что в этом массиве действительно есть проверяемое k. Итоговая асимптотика — $O((n^3 + q) \log n)$.

Чтобы решить 3-ю и 4-ю подзадачи, давайте оптимизируем решение из 1-й подгруппы. Заметим, что мы можем перебирать разбиение на 2 части, начиная с конца подотрезка. Таким образом, мы можем поддерживать как произведение элементов в правой части, так и сумму в левой части. Соответственно, на каждый запрос мы будем тратить $O(n)$, а итоговая асимптотика будет равна $O(qn)$.

Теперь давайте обсудим решения 5-й и 6-й подзадачи. Ограничение на то, что a_i чётное, означает, что a_i всегда должно быть больше 1. На самом деле это означает, что нам достаточно проверить не более $O(\log k)$ возможных разбиений, так как все разбиения, в которых в правой части больше $\log k$ элементов, заведомо нам не подойдут, поскольку произведение элементов будет больше k. Единственное, что нам осталось сделать, — это как-то поддерживать сумму в левой части. В случае 5-й задачи нам достаточно использовать префиксные суммы, так как у нас нет запросов на изменения. В 6-й подзадаче для этой цели придётся использовать дерево отрезков либо дерево Фенвика. Асимптотика в таком случае будет $O(n \log n + q(\log k + \log n))$.

Чтобы решить последние две подзадачи, сделаем следующее замечание: в правой части нужного нам разбиения может быть не более $\log k$ элементов, больших 1. В таком случае давайте просто перебирать количество элементов, больших 1, в правой части, поддерживая сумму в левой части, и на каждом шаге проверять, можем ли мы как-то поделить подотрезок между текущим неединичным элементом и следующим так, чтобы произведение частей было равно k. Чтобы это реализовать в 7-й подзадаче, нам потребуется просто предпосчитать для каждого неединичного элемента ближайший элемент слева, больший 1, а в 8-й подзадаче для этого можно реализовать дерево отрезков. Одна из возможных реализаций имеет асимптотику $O(n \log n + q \log k \log n)$.

Задача B. Гонки

Для решения первой подзадачи достаточно перебрать все возможные отрезки длины хотя бы k. Для каждого отрезка можно найти сумму путём сложения всех элементов и получить среднее арифметическое. Всего таких отрезков будет $O(n^2)$ и общая сложность решения — $O(n^3)$.

Для второй подзадачи следует воспользоваться алгоритмом префикс-сумм, чтобы находить сумму на отрезке за $O(1)$. Тогда общая сложность алгоритма будет $O(n^2)$.

Для полного решения можно заметить следующий факт: всегда существует оптимальный ответ, в котором длина отрезка менее $2 \cdot k$. Действительно, рассмотрим какой-нибудь отрезок длины хотя бы $2 \cdot k$, пусть среднее арифметическое его элементов равно d. Этот отрезок можно разбить на два отрезка длины хотя бы k, при этом, если у обоих отрезков среднее арифметическое меньше d, то и среднее арифметическое исходного отрезка было бы меньше d (это легко доказать). Значит, хотя бы один из отрезков меньшей длины даёт ответ не хуже. Значит, самый короткий из оптимальных отрезков имеет длину менее $2 \cdot k$. Теперь осталось перебрать длину этого отрезка в интервале от k до $2 \cdot k - 1$ и для каждой длины найти наибольшее возможное среднее арифметическое за $O(n)$ операций при помощи префикс-сумм. Общая сложность такого решения будет $O(n \cdot k)$.

Замечание 1. Эту задачу можно решить при помощи бинарного поиска по ответу с асимптотикой $O(n \cdot log(n))$. При этом не потребуется идея о длине оптимального отрезка, решение не зависит

от k.

Замечание 2. Для сравнения средних арифметических значений двух отрезков рекомендуется вычислять эти значения в виде дробей. Далее для сравнения дробей можно использовать либо 128-битный тип данных, либо сначала сравнить целые части, после чего останутся две дроби, каждое из чисел в которых не превосходит n. Эти дроби уже можно сравнить, используя 64-битные целые типы.

Задача C. Студент и электрички

Построим взвешенный неориентированный граф G, вершины которого — города, и между парой городов проведено ребро веса w, если существует электричка уровня w, которая останавливается в каждом из них. Тогда требуется по запросу (c, d) найти вес минимального пути из c в d, где вес пути — максимум из всех весов пройденных рёбер. Значит, достаточно построить минимальное остовное дерево в G (например, с помощью алгоритма Краскала) и найти вес пути в этом дереве. При этом, для каждой электрички достаточно провести только рёбра между соседними остановками, поэтому в полученном графе G не более nm рёбер и n вершин. Таким образом, можно отвечать на запрос за $O(n)$ с предподсчётом за $O(nm \cdot \alpha(nm))$. Такое решение проходит первые три подгруппы.

Для решения 4-й и 5-й подзадач заметим, что каждое ребро графа G проходит между городами i и $i + 1$. Тогда достаточно для каждой пары соседних городов i и $i + 1$ найти минимальный уровень электрички, проходящей между ними. Для этого можно воспользоваться, например, деревом отрезков (присвоить на отрезке $[l_j, r_j]$ значение w_j для всех электричек в порядке убывания уровня), или методом сканирующей прямой (на i-м шаге поддерживать множество отрезков, таких, что $l_j \leqslant i \leqslant r_j$).

Для полного решения также воспользуемся алгоритмом Краскала. Будем поддерживать систему непересекающихся множеств (изначально каждый город поместим в отдельное множество) и дерево отрезков с изменением в точке и следующей операцией: найти на отрезке $[l, r)$ пару городов (i, j) из разных множеств такую, что пара (h_i, h_j) лексикографически минимальна (или сообщить, что такой пары не существует). Для того, чтобы найти оптимальную пару на отрезке, достаточно знать лишь соответствующие пары для его детей. Тогда, обрабатывая электрички в порядке возрастания уровня, для каждой i-й электрички выполним запрос в дереве отрезков и, возможно, добавим в дерево рёбра веса w_i. Если оказалось, что подходящей пары не существует или высота платформы в одном из городов больше максимально допустимой для данной электрички, то все города, где останавливается i-я электричка, уже соединены рёбрами веса не более w_i, — тогда перейдём к следующей электричке. Иначе соединим найденную пару городов ребром веса w_i, объединим соответствующие множества и повторим запрос. При этом, при объединении множеств необходимо обновить некоторые значения в дереве отрезков, поэтому будем поддерживать набор элементов в каждом множестве и перемещать элементы из меньшего множества в большее — тогда всего произойдёт $O(n \log n)$ перемещений. При перемещении элемента обновим соответствующую вершину в дереве отрезков. Таким образом, построим искомое остовное дерево за $O(n \log^2 n + m \log n)$. Далее для ответа на запрос достаточно найти максимальный вес ребра на пути между заданными двумя вершинами дерева — это можно сделать за $O(\log n)$ с помощью, например, двоичных подъёмов. Таким образом, сложность решения — $O(n \log^2 n + (m + q) \log n)$.

Задача D. Бег с препятствиями

В задаче нужно найти первую точку X, такую что на обеих полосах в этой точке будет препятствие. Тогда ответом будет $X - 1$.

Подгруппа 1

При $k = 0$ препятствия находятся на первой полосе в точках $a,\ 2a,\ 3a,\ ...$, на второй $b,\ 2b,\ 3b,\ ...$

На обеих полосах будет препятствие, если $x \cdot a = y \cdot b$, а самая первая такая точка — $lcm(a, b)$ (наименьшее общее кратное).

Ответом будет $lcm(a, b) - 1$.

Решение $O(log(a, b))$.

Подгруппа 2

Если $a = 2$, то $k = 0$ или $k = 1$.

При $k = 0$ на первой полосе препятствия на каждой четной точке. Значит, если на второй полосе препятствие будет тоже на четной точке, то дальше продвинуться нельзя.

Ответ $b - 1$, если b изначально четное и $2 \cdot b - 1$, если нечетное. Исключение при $b = 1$, тогда ответ будет 1, так как можно будет добраться до точки 1 на первой полосе.

При $k = 1$ препятствия на первой полосе будут на всех точках, кроме 0.

Значит, двигаться можно строго по второй полосе и остановимся перед первым препятствием, то есть в точке $b - 2$.

Решение $O(1)$.

Подгруппа 3

Будем перебирать каждую точку до n включительно и проверять, на обеих ли полосах есть препятствие или нет.

Пусть $x = 1$ — коэффициент для a. При переборе всех точек i проверяем, что точка лежит на отрезке $[x \cdot a - k,\ \ldots x \cdot a]$, если это так, значит на точке i на первой полосе есть препятствие. А если $i > x \cdot a$, то увеличиваем x на 1, чтобы перейти к следующему блоку препятствий. Аналогично делаем для второй полосы, таким образом сможем найти точку, где на обеих полосах будет препятствие.

Решение $O(n)$.

Полное решение

Перебирать каждую точку — долго, нужно сразу перебирать блоки препятствий. Таких блоков будет порядка $\frac{n}{a}$ на первой полосе и $\frac{n}{b}$ на второй.

При этом нужная точка находится не дальше, чем $a \cdot b$. То есть для первой полосы перебираем $min(\frac{n}{a}, b)$ блоков, а для второй $min(\frac{n}{b}, a)$ блоков. Значит, перебирая одновременно коэффициенты для первой и второй полосы, мы найдем ответ за $min(\frac{n}{a}, b, \frac{n}{b}, a) \leqslant \sqrt{n}$.

Решение $O(\sqrt{n})$.

Задача E. Просто песня

Для начала нужно удалить из песни все пробелы, ибо нас будут интересовать только ударные и не ударные буквы.

Первые две подзадачи можно решить при помощи динамического программирования. Пусть $dp[p]$ — количество способов правильно продлить последовательность слов, начиная с позиции p в песне ($p \leqslant s_m$). Здесь можно перебрать каждое из n слов и попробовать подставить его, начиная с позиции p. Если оно накроет только одну ударную букву в песне и позиция этой буквы совпадет с позицией ударной буквы в слове, то ответ надо увеличить на $dp[p + k_i]$, где k_i — длина очередного слова Эдуарда.

Для ускорения этого решения можно перебирать только те слова, ударение в которых расположено так, что при записи слова с позиции p в песне, ударения в песне и в слове совпадут. Для этого каждое слово заменим на пару (l_i, r_i) — где l_i показывает количество букв перед ударением в слове, а r_i — количество букв после ударения. Далее для каждой позиции p нужно преподсчитать позицию ближайшей справа ударной буквы — $next_p$. Теперь для каждой позиции p мы знаем количество букв без ударения, с которых должно начинаться слово (это величина $next_p - p$) и максимальное количество букв, которое может идти после ударения — это расстояние между следующими двумя ударными буквами. Таким образом, мы знаем, чему должно равняться l_i и ограничение сверху на r_i, что позволит перебирать только подходящие слова.

Такой подход можно еще ускорить — для каждого l будем хранить все такие r, что существует хотя бы одно слово, для которого $l_i = l$ и $r_i = r$. Более того, будем хранить не только список значений r, но и количество слов, у которых l_i и r_i принимают такие значения. Теперь при переборе подходящих слов все значения r_i будут различны, а значение $dp[p + k_i]$ нужно умножить на количество соответствующих слов в словаре Эдуарда.

Оказывается, это решение работает за $O(max(s_n, s_m)^{\frac{4}{3}})$. Действительно, для каждой позиции ударной буквы в песне аналогично посчитаем L_i и R_i — количество не ударных букв слева и справа. Теперь для каждой позиции ударной буквы найдем, сколько раз во время работы описанного выше алгоритма мы ставили слово, у которого ударная буква попадает на эту позицию. Очевидно, количество таких операций не превосходит $(L_i + 1) \cdot (R_i + 1)$ — для каждого $0 \leqslant l \leqslant L_i$ и $0 \leqslant r \leqslant R_i$ мы рассматриваем пару (l, r) максимум один раз для данной позиции. Теперь разобьем все такие

пары на две группы. К первой отнесем те слова, где $l \leqslant max(sn, sm)^{\frac{1}{3}}$. В сумме для всех таких слов количество операций будет не больше $\sum l \cdot (R_i + 1) \leqslant sm^{\frac{1}{3}} \cdot \sum(R_i + 1) \leqslant sm^{\frac{1}{3}} \cdot sm = sm^{\frac{4}{3}}$. Теперь рассмотрим случай, когда $l > max(sm, sn)^{\frac{1}{3}}$. Каждое такое слово из словаря Эдуарда будет содержать хотя бы $sn^{\frac{1}{3}}$ букв, значит, таких слов не более $sn^{\frac{2}{3}}$. При этом, количество позиций, при которых $L_i > sm^{\frac{1}{3}}$ не будет превышать $sm^{\frac{2}{3}}$, то есть общее число операций будет не более $sn^{\frac{2}{3}} \cdot sm^{\frac{2}{3}} \leqslant max(s_n, s_m)^{\frac{4}{3}}$.

Задача F. Достойное продолжение

Для решения первых двух подгрупп достаточно перебрать все пары чисел и для каждой проверить, что первая цифра второго числа совпадает с последней цифрой первого. При этом для первой подгруппы ($a_i \leqslant 99$) легко получить первую и последнюю цифры простым делением на 10. Во второй подгруппе придется либо делить на 10 в цикле, либо хранить числа как строки. Сложность решения — $O(n^2)$.

Для решения полной версии задачи можно для каждой цифры d посчитать, сколько чисел начинаются с этой цифры — s_d и сколько чисел заканчиваются на неё — t_d. Тогда для нахождения ответа достаточно сложить значения $s_d \cdot t_d$ для каждой цифры. Единственное, что нужно учесть, при таком решении будут считаться и пары одинаковых чисел (когда число дописывается к самому себе) — для каждого числа, у которого первая цифра совпадает с последней, ответ надо уменьшить на 1. Итого, сложность решения есть $O(n)$.

Задача G. Максимальный XOR

Подгруппа 1

Переберем все пары i, j такие, что $i \neq j$ и для каждой пары пересчитаем XOR на массиве, если бы вместо a_i было $a_i + x$, вместо a_j было $a_j + y$. Перебор за n^2 и еще n на пересчет XOR на массиве.

Решение $O(n^3)$.

Подгруппа 2

Аналогично перебираем пары i, j такие, что $i \neq j$. Но перед этим изначально запомним XOR на всем массиве.

Тогда, если мы хотим поменять a_i на $a_i + x$, то чтобы пересчитать XOR — нужно сделать следующее XOR = (XOR $\oplus$ a_i) $\oplus$ $(a_i + x)$. Так как для любого числа $A \oplus A = 0$.

Таким образом, если мы меняем одно число на другое, то пересчитать общий XOR можно за одну операцию, а для каждой пары нам нужно сделать по 2 такие операции.

Решение $O(n^2)$.

Подгруппа 3

$x = 0$ равносильно тому, что мы меняем только одно число, а не два. Поэтому достаточно перебрать каждый элемент и попытаться заменить его на $a_j = a_j + y$ и среди ответов найти максимальный.

Решение $O(n)$.

Подгруппа 4

Все числа в массиве до 1000, можно посчитать, сколько раз каждое число использовалось, например, завести мапу или обычный массив счетчиков.

Пусть cnt_i — количество чисел i в изначальном массиве.

Переберем все пары $cnt_i > 0, cnt_j > 0$, здесь $i \neq j$ и заменим в общем XOR i на $i + x$, j на $j + y$, то есть выполним следующую операцию XOR = (XOR $\oplus$ i) $\oplus$ $(i + x)$, затем XOR = (XOR $\oplus$ j) $\oplus$ $(j + y)$.

Остается перебрать все $cnt_i > 1$, так как к двум одинаковым числам мы могли прибавить к первому x, а ко второму y, таким же образом пересчитаем общий XOR.

Решение $O(max\_element(a)^2)$.

Полное решение

Пусть $X_i = a_i \oplus (a_i \oplus x)$, $Y_i = a_i \oplus (a_i \oplus y)$, xr — общий XOR изначального массива.

Тогда, если мы меняем $a_i = a_i + x$, $a_j = a_j + y$, то $new\_xr = xr \oplus X_i \oplus Y_j$. Давайте введем $Z_j = xr \oplus Y_j$, тогда $new\_xr = X_i \oplus Z_j$. Таким образом, задача сводится к поиску двух элементов X_i, Z_j с различными индексами, дающих максимальное значение XOR.

Предпосчитать все массивы X, Y, Z можно за один проход.

Давайте построим битовый бор по всем X_i, так как числа $a_i \leqslant 10^9 < 2^{31}$, то вставка элемента в бор будет работать за $O(32) = O(1)$.

Аналогично за $O(32) = O(1)$ в боре мы сможем удалить какой-то элемент и найти какое-то число, которое дает максимальный XOR с заданным Z_j. Остается только перебрать эти Z_j. Еще стоит не забыть, перед каждым запросом для Z_j нужно удалить X_j из бора, чтобы случайно не прибавить y к тому же числу, к которому изначально добавили x, а после запроса не забыть вернуть X_j в бор.

Решение $O(n)$.

Задача H. Фёдор и ферзи

Для решения первых групп тестов нужно реализовать рекурсивный перебор, который будет рассматривать все возможные позиции ферзей и выбирать оптимальную. При этом для ускорения работы данного перебора и прохождения более сложных тестов можно использовать следующие приемы:

- перебирать ферзей в лексикографическом порядке, то есть позиция каждого следующего ферзя должна быть больше (больше номер строки, а при равенстве — больше номер столбца) предыдущего. Это позволит сократить перебор в $k!$ раз;
- не ставить ферзей на уже занятые строки или столбцы. Для данных ограничений это все равно позволит найти верный ответ. Однако, если ставить только не бьющих друг друга ферзей, то на некоторые тесты ответ будет неверным (например, для $n = 12$ и $k = 6$ таким образом можно покрыть только 142 клетки, в то время как 6 ферзями можно покрыть всю доску 12×12);
- в силу симметрии доски относительно вертикали, горизонтали и диагоналей, можно считать, что первый ферзь расположен наиболее близко к углу доски (левому-верхнему). Это сократит перебор еще в 8 раз;
- поддерживать количество защищенных клеток в момент постановки очередного ферзя (вместо пересчета после постановки всех ферзей);
- добавить отсечение по ответу — если покрыта вся доска, завершить перебор. Если оставшимися ферзями нельзя улучшить ответ, выйти из этой ветки рекурсии. Можно придумать несколько разных оценок количества клеток, которые смогут защитить оставшиеся ферзи. Например, если остался один ферзь, то он дополнительно покроет не более $4 \cdot (n-k)+1$ клеток, поскольку $(k-1)$ строка и столбец уже полностью защищены;

Для прохождения более сильных тестов, где $n > 12$, можно воспользоваться предподсчетом — запустить перебор локально и сохранить ответ для каждой возможной пары n, k. За разумное время ($1-10$ минут) эффективная реализация перебора с применением техник, описанных выше, позволит найти ответы практически на все тесты.

Для самых сложных случаев — когда $k = 8$ — перебор может работать слишком долго даже локально. Здесь будет уместно использовать метод отжига или другие популярные методы для задач комбинаторной оптимизации вместе с локальным предподсчетом.