

Задача А. Задача из Магнитогорска

Разработчик задачи: Егор Зарубин

В этой задаче можно было перебрать все перестановки цифр числа a , проверить только те, в которых нет ведущих нулей, и сравнить с числом b . Так как числа во входных данных не больше 999, решение можно реализовать за $O(1)$ для одного набора входных данных.

Задача В. Задача из Карабаша

Разработчик задачи: Мария Ткаченко

В этой задаче нужно было рассмотреть два случая отдельно:

- $a = 1$. В этом случае все стоимости такси будут вида $x_i = i \cdot b$ и достаточно проверить, что между числами l и r есть число, которое делится на b . Для этого достаточно проверить, что $\lfloor \frac{r}{b} \rfloor \neq \lfloor \frac{l-1}{b} \rfloor$.
- $a > 1$. В этом случае можно с помощью цикла найти все стоимости такси, которые не больше r . Их будет не более $O(\log_a r)$, поэтому решение будет работать быстро.

Задача С. Задача из Новогорного

Автор и разработчик задачи: Екатерина Шилева

Крайним случаем в этой задаче является $n = 1$, так как никакую перестановку нельзя разбить на два непустых подотрезка. Ответ в этом случае равен 0.

Иначе представим перестановку как два возрастающих подотрезка. Рассмотрим все варианты распределения чисел по этим двум возрастающим подотрезкам. Всего таких вариантов будет 2^n — каждый элемент мы можем определить в один из подотрезков.

Заметим, что все такие разбиения приведут к разным перестановкам, кроме случаев, где все числа первого подотрезка меньше всех чисел второго подотрезка — такие разбиения формируют перестановку $(1, 2, \dots, n)$ (в возрастающем порядке). Такая перестановка будет учтена $n + 1$ раз.

Из всего этого следует, что для $n \geq 2$ ответ равен $2^n - n$. Так как n достаточно большое, для нахождения ответа нужно воспользоваться быстрым возведением в степень.

Решение можно реализовать за $O(\log n)$.

Задача D. Задача из Миасса

Автор и разработчик задачи: Илья Богатырев

Будем строить набор максимально возможных глубин, то есть такой, что -1 в каждой ячейке заменяется на наибольшее число, не нарушающее при этом никаких условий задачи.

Заметим, что каждое известное (не равное -1) значение глубины d_{ij} в некоторой ячейке (i, j) порождает множество ограничений на максимальную глубину во всех других ячейках. А именно, ячейки, находящиеся от (i, j) на манхэттенском расстоянии, равном 1, не могут иметь глубину больше, чем $d_{ij} + v$. Ячейки, находящиеся на манхэттенском расстоянии 2, не могут иметь глубину больше, чем $d_{ij} + 2 \cdot v$, и так далее. Если глубина в некоторой ячейке нарушает хотя бы одно из ограничений, то она не может являться частью корректного ответа. То есть, значение в любой ячейке должно удовлетворять наиболее строгому (минимальному по величине) из ограничений в ней. Обозначим его $dmax_{ij}$.

Покажем, что значения $dmax_{ij}$ в любых соседних ячейках не могут отличаться больше, чем на v . Предположим обратное: нашлась какая-то пара значений d_1 и d_2 , такая что $d_1 > d_2 + v$. Поскольку d_2 — это просто минимум из ограничений в ячейке, то существует какое-то ограничение, на котором этот минимум достигается. Рассмотрим ограничения на соседние ячейки, порожденные тем же значением d_{ij} . Поскольку манхэттенское расстояние до соседних клеток отличается ровно на 1, то эти ограничения могут быть равны $d_2 + v$ или $d_2 - v$. В любом случае, имеем значения не больше $d_2 + v$.

А значит, и минимум из всех ограничений в любой из соседних клеток не может превосходить этой величины. Получили противоречие с тем, что $d_1 > d_2 + v$. Значит, построенный таким образом набор глубин $dmax_{ij}$ удовлетворяет условию задачи: разность любых соседних значений не превосходит v .

Заметим, что рассуждение выше верно только в том случае, если все уже известные глубины удовлетворяют всем ограничениям, порожденным другими глубинами. Если же в некоторой ячейке $d_{ij} > dmax_{ij}$, то существуют какие-то две ячейки, находящиеся на манхэттенском расстоянии x и имеющие разность глубин больше, чем $v \cdot x$. В этом случае построить требуемое невозможно.

Покажем, как можно эффективно вычислять $dmax_{ij}$. Разобьем все ограничения, существующие в ячейке (i, j) , на следующие группы:

- ограничения, порожденные значениями в ячейках слева сверху от (i, j) , то есть такими, что их индексы (i', j') удовлетворяют условиям $i' \leq i, j' \leq j$;
- ограничения, порожденные значениями справа сверху от (i, j) ($i' \leq i, j' \geq j$);
- ограничения, порожденные значениями слева снизу от (i, j) ($i' \geq i, j' \leq j$);
- ограничения, порожденные значениями справа снизу от (i, j) ($i' \geq i, j' \geq j$).

Очевидно, что любое из ограничений попадает как минимум в одну из групп.

Соответственно, запустим несколько отдельных проходов по таблице: слева сверху, справа сверху, слева снизу и справа снизу. При каждом проходе будем вычислять $dmax_{ij}$ с учетом ограничений только выбранной группы. Значения пересчитываются на основе значений соседей, уже вычисленных ранее в том же проходе. Например, при проходе слева сверху:

$$dmax[i][j] = \min(dmax[i-1][j] + v, dmax[i][j-1] + v).$$

Если хотя бы в одном из проходов в любой из ячеек мы получим $d_{ij} > dmax_{ij}$, то ответ к задаче «No». В противном случае (противоречий не обнаружилось) — «Yes».

Задача Е. Задача из Депутатского

Автор и разработчик задачи: Никита Зибницкий

Заметим, что, зная текущую позицию одного человека и начальные позиции обоих, можно однозначно восстановить текущую позицию второго. Пусть начальные позиции равны (a_0, b_0) и (c_0, d_0) , а текущая позиция первого — (a_1, b_1) . Тогда текущая позиция второго равна

$$((c_0 + a_0 - a_1) \bmod n, (d_0 + b_0 - b_1) \bmod m).$$

Отсюда следует, что существует не более $n \cdot m$ различных пар состояний $((a, b), (c, d))$, в которых оба человека могут одновременно находиться. Поэтому достаточно построить граф этих состояний и запустить по нему обход в ширину (bfs), проверяя достижимость любого состояния вида $((a, b), (a, b))$ (то есть совпадения позиций). Поскольку из каждой вершины графа исходит не более 4 переходов, время работы и требуемая память составляют $O(n \cdot m)$.

Задача F. Задача из Копейска

Автор задачи: Илья Богатырев. Разработчик задачи: Никита Насонков

Рассмотрим два целых положительных числа x и y . Пусть они раскладываются на простые множители следующим образом: $x = p_1^{k_1} \cdot p_2^{k_2} \cdot \dots \cdot p_c^{k_c}$ и $y = p_1^{m_1} \cdot p_2^{m_2} \cdot \dots \cdot p_c^{m_c}$. Тогда, чтобы x было кратно y , необходимо и достаточно, чтобы для каждого простого множителя в разложении y его степень была меньше или равна степени этого множителя в разложении x . Формально, $\forall i \in [1, c] \ k_i \geq m_i$.

Значение $f(x)$ из условия задачи должно быть кратно каждому числу от 1 до x включительно по определению наименьшего общего кратного. Получается, если в разложении на простые множители $f(x) = p_1^{k_1} \cdot p_2^{k_2} \cdot \dots \cdot p_c^{k_c}$ и $y = p_1^{m_1} \cdot p_2^{m_2} \cdot \dots \cdot p_c^{m_c}$, то $\forall y \in [1, x] \ \forall i \in [1, c] \ k_i \geq m_i$, иными словами,

степень каждого простого числа в разложении $f(x)$ на простые множители должна быть больше или равна максимальной степени этого простого числа, встречающейся в разложениях чисел от 1 до x (то есть префиксному максимуму степени этого простого числа в разложении чисел от 1 до x). Поскольку $f(x)$ — это **наименьшее** общее кратное, то мы должны взять минимально возможное k_i для каждого i , поэтому $\forall i \in [1, c] \ k_i = m_i$.

Теперь рассмотрим ряд $f(1), f(2), f(3), \dots, f(x), \dots$. Заметим, что значения в нём не убывают, так как, как мы уже выяснили, $f(x)$ зависит от префиксных максимумов степеней простых чисел в разложении, а префиксные максимумы не убывают.

Для начала решим получившуюся подзадачу нашей задачи: есть неубывающая последовательность чисел, нужно быстро находить количество различных чисел на отрезке $[l, r]$. Для этого нам нужно найти все индексы i такие, что $f(i) > f(i-1)$. Сформируем массив a такой, что $a[i] = 1$, если $i = 1$ или $f(i) > f(i-1)$, иначе $a[i] = 0$ (о том, как сформировать такой массив, будет написано ниже). Иными словами, единицы в массиве a будут в тех позициях, где $f(i)$ меняет своё значение по сравнению с предыдущим. Тогда количество различных чисел на отрезке $[l, r]$ будет равно количеству единиц на отрезке $[l, r]$ в массиве a (возможно, $+1$). Почему так? Пройдёмся слева направо по отрезку $[l, r]$ в массиве a . Если $a[i] = 1$, то это новое значение $f(i)$, добавим 1 к ответу. Иначе $a[i] = 0$, значит, такое значение $f(i)$ уже встречалось, и в массиве a где-то ранее была единица, которую мы уже прибавили к ответу, поэтому ничего делать не нужно. Исключение из этого правила — это позиция $a[l]$, в которой может быть 0, для которого мы не прибавили к ответу единичку (так как она была левее отрезка $[l, r]$). Поэтому если $a[l] = 0$, то к количеству различных чисел на отрезке надо прибавить 1. Количество единиц в массиве a на отрезке можно посчитать при помощи префиксных сумм за $\mathcal{O}(1)$ с предподсчётом за $\mathcal{O}(maxr)$ ($maxr$ в задаче равно 10^7). В итоге мы научились быстро (за $\mathcal{O}(1)$) отвечать на запрос количества различных чисел на отрезке $[l, r]$, имея массив a .

Теперь поговорим о том, как получить вышеупомянутый массив a . Для этого можно воспользоваться решето Эратосфена. Сначала присвоим $a[1] = 1$. Затем, когда мы в процессе выполнения алгоритма нашли очередное простое число p , то во внутреннем цикле, когда мы проходимся по всем числам, кратным p , чтобы пометить их как составные, нужно отдельно для каждого числа z , равного p^k , где k — целое положительное число и $k \geq 1$, присвоить $a[z] = 1$. Таким образом, $a[i]$ станет равно 1 в тех индексах i , где увеличивается префиксный максимум степени какого-либо простого числа из разложения на простые множители, а значит, как мы уже выяснили, увеличивается и значение функции f — такой массив a нам и нужен! Это всё нужно проделать один раз до ответа на запросы. После этого мы получим массив a , по которому построим массив префиксных сумм и сможем отвечать на запросы за $\mathcal{O}(1)$.

Итого наш алгоритм выполняет решето Эратосфена за $\mathcal{O}(maxr \cdot \log \log maxr)$, формирует массив префиксных сумм за $\mathcal{O}(maxr)$, а затем отвечает на t запросов за $\mathcal{O}(1)$ на каждый. Итоговое время работы составит $\mathcal{O}(maxr \cdot \log \log maxr + t)$.

Задача Г. Задача из Берлина

Автор задачи: Екатерина Шилева. Разработчик задачи: Арсений Ятчений

По условию максимальная стоимость покраски не превышает 100. Тогда рассмотрим такую пару длин len_1, len_2 , что $1 \leq len_1 < len_2 \leq n$ и $cost_{len_1} = cost_{len_2}$, тогда за стоимость покраски подотрезка длины len_1 мы можем покрасить подотрезок длины len_2 , поэтому среди пары длин подотрезков с одинаковой стоимостью всегда выгоднее брать ту, у которой длина больше. Тогда у нас остается не более 100 *интересных* длин, которые мы будем рассматривать для подсчета динамики.

Чтобы быстро проверять, есть ли хотя бы одна 1 на отрезке $[l, r]$ в изначальном массиве, в самом начале предподсчитаем массив префиксных $pref_i = \sum_{j=1}^i a_j$ и тогда для запроса проверки наличия единицы на отрезке $[l, r]$ нужно проверить, что $pref_r - pref_{l-1} > 0$.

Будем считать динамику dp_i — минимальная сумма, за которую можно покрасить префикс длины i . На самом деле это будет не совсем так — если на префиксе не будет единиц, то мы тоже посчитаем динамику, просто будет подразумеваться, что мы до этой единицы доберёмся позже.

- База динамики: $dp_0 = 0$

- Переходы выглядят следующим образом:

Пусть мы хотим пересчитать dp_i . Будем перебирать длину подотрезка, который мы хотим сейчас покрасить (покрасить последним), пусть она равна len .

Тогда возможны два случая:

- На подотрезке $[i - len + 1, i]$ в изначальном массиве a есть хотя бы одна единица. Тогда мы можем за $cost_{len}$ покрасить подотрезок $[i - len + 1, i]$ и соответственно пересчитать dp_i через $dp_{i-len} + cost$
- На подотрезке $[i - len + 1, i]$ в изначальном массиве a нет единиц. Тогда мы должны потребовать, чтобы мы пересекались с другим подотрезком, и в пересечении находилась хотя бы одна единица. Таким образом, в данном случае dp_i пересчитывается через $dp_{i-len+1} + cost_{len}$.

Так как мы рассматриваем всего 100 длин и проверка наличия на отрезке единицы выполняется за $\mathcal{O}(1)$, то итоговое время работы будет равно $\mathcal{O}(100 \cdot n)$ для одного набора входных данных.

Задача Н. Задача из Златоуста

Разработчик задачи: Глеб Костылев

Будем считать, что $n \leq m$. Если это не так, то мы можем повернуть матрицу на 90 градусов.

Ответ на задачу «Yes» тогда и только тогда, когда в некоторой строке матрицы существует возрастающая подпоследовательность длины хотя бы $m - n + 1$. Докажем это. Если в строке i существует такая последовательность с индексами $1 \leq j_1 < j_2 < \dots < j_{m-n+1} \leq m$ (то есть, $a_{i,j_1}, a_{i,j_2}, \dots$ образуют возрастающую последовательность), то мы можем удалить все строки, кроме i -й, и все столбцы, кроме j_k -х, и оставшаяся матрица будет удовлетворять условию. Если же такой подпоследовательности не существует, то мы не сможем добиться желаемого. Какие бы столбцы и строки мы ни удаляли, в результате у нас останется некоторая подматрица исходной матрицы с хотя бы одной строкой и хотя бы $m - n + 1$ столбцами. Раз мы предположили, что возрастающей подпоследовательности длины $m - n + 1$ нет ни в какой строке матрицы, то строки полученной подматрицы не отсортированы, а значит, добиться ответа «Yes» нельзя вне зависимости от наших действий.

Теперь нам остаётся найти длину наибольшей возрастающей подпоследовательности (НВП) в каждой строке матрицы и проверить, что хотя бы в какой-то строке она имеет длину хотя бы $m - n + 1$. Поиск НВП — стандартная задача, решить её можно при помощи дерева отрезков, дерева Фенвика или динамического программирования за $\mathcal{O}(m \log m)$ для каждой строки.

Итого решение можно реализовать за $\mathcal{O}(nm \log \max(n, m))$.

Задача I. Задача из Малиновки

Автор и разработчик задачи: Екатерина Шилева

Заметим, что для построения ответа неважно, в каком порядке выполнять желания Бараша, важно лишь соблюдать ограничение на количество конфет каждого типа. Будем сначала выполнять все желания, где нужны конфеты одного типа. Для этого будем каждый раз брать по две конфеты того типа, где осталось наибольшее количество конфет. Если в какой-то момент нельзя взять две конфеты таким образом или кончатся дни, когда нужно выбрать две одинаковые конфеты, перестанем это делать и перейдем к выполнению желаний на разные конфеты.

Для выполнения желаний на разные конфеты будем каждый раз брать одну конфету вида с максимальным количеством оставшихся конфет и одну конфету вида с минимальным оставшимся количеством конфет, пока можно так сделать.

После проведения таких операций в оставшиеся дни выберем любые две конфеты из оставшихся и посчитаем ответ, зная количество всех дней, для которых были выполнены желания. Можно показать, что такие операции приведут к оптимальному ответу.

Не умаляя общности, предположим, что $a \leq b \leq c$. Заметим, что выполнить все желания для разных конфет можно лишь при условии, что $a + b \geq c$, а в случае, если это условие не выполняется, можно выполнить лишь $a + b$ таких желаний. Поэтому когда мы сначала рассматриваем желания одинаковых конфет и выбираем две конфеты, которых осталось больше всего, мы приближаемся к случаю, в котором $a + b \geq c$.

Решение можно реализовать за $\mathcal{O}(n)$.

Задача J. Задача из Челябинска

Автор и разработчик задачи: Екатерина Шилева

Заметим, что когда мы ставим чёрную дыру в какую-то из галактик (в какой-то подотрезок), сумма на нём больше не влияет на массу Вселенной (мы можем предполагать, что масса чёрной дыры бесконечно маленькая). Сделаем бинарный поиск по ответу и будем проверять, можно ли получить массу Вселенной не более чем текущая середина.

Для проверки этого необходимо лишь узнать, можно ли «выкинуть» из рассмотрения в массе Вселенной все галактики с массой больше текущей середины бинарного поиска. Для подсчета суммы на каждом подотрезке можно воспользоваться префиксными суммами. Так как «выкинуть» подотрезок можно лишь предположив на нём где-то чёрную дыру, и нам необходимо поставить чёрную дыру в каждый из подотрезков с суммой больше текущей середины, мы сможем это сделать, только если все такие подотрезки имеют непустое пересечение в каком-то индексе массива. То есть, необходимо просто найти пересечение всех галактик с изначальной суммой больше, чем текущая середина бинарного поиска, и проверить, что они пересекаются (это можно сделать, поддерживая левую и правую границу текущего пересечения и обновляя их на очередном подотрезке).

Если все галактики имеют непустое пересечение, то можно сделать сколь угодно малую массу Вселенной, предположив чёрную дыру в любом индексе пересечения.

Решение можно реализовать за $\mathcal{O}(n \cdot \log(n \cdot \max(|a_i|)))$.

Задача K. Задача из Озёрска

Автор и разработчик задачи: Иван Добрынин

Рассмотрим случай, когда $n = 1$, то есть мы можем двигаться только в пределах одной плитки. Представим поле из $h \times w$ белых и чёрных клеток в виде неориентированного графа. Вершинами в нём будут белые клетки, а рёбра проведём между соседними клетками. В таком графе выделим компоненты связности. Пусть $comp_{r,c}$ — номер компоненты связности клетки (r, c) . Для ответа на запрос будет достаточно сравнить $comp_{r_1, c_1}$ и $comp_{r_2, c_2}$.

При переходе к $n > 1$ добавляются переходы между плитками. Пронумеруем плитки слева направо числами от 1 до n . Пусть клетка (r_1, c_1) лежит в плитке g_1 , а клетка (r_2, c_2) — в плитке g_2 . Если $g_1 > g_2$, то ответ «No», так как зазор между плитками можно проходить только в направлении слева направо. Если $g_1 = g_2$, то в этом случае мы можем двигаться только в пределах одной плитки, и здесь будет достаточно решения из случая $n = 1$.

Рассмотрим переход через зазор между двумя плитками. Мы попадаем из самого правого столбца одной плитки в самый левый столбец второй плитки. Пусть переход происходит в строке r . Тогда мы попадаем из компоненты связности $comp_{r, h-1}$ в компоненту связности $comp_{r, 0}$, если обе клетки $(r, h-1)$ и $(r, 0)$ — белые. На самом деле, задача свелась к тому, чтобы определить, можно ли ровно за $g_2 - g_1$ переходов через зазор между двумя плитками попасть из $comp_{r_1, c_1}$ в $comp_{r_2, c_2}$.

Построим ориентированный граф, вершинами в котором будут компоненты связности изначального графа, а рёбра проведём в соответствии с переходами через зазор между двумя плитками. В таком графе $\mathcal{O}(h)$ вершин будут иметь хотя бы одно входящее или исходящее ребро, а рёбер будет не больше h , так как для каждой строки плитки мы проводим максимум одно ребро.

Рассмотрим подробнее, как выглядит этот ориентированный граф. Во-первых, в нём могут быть петли, что равнозначно возможности сколь угодно долго оставаться в одной и той же компоненте связности при переходе между плитками. Во-вторых, подумаем, могут ли в этом графе существовать циклы, не являющиеся петлями. Докажем от противного, что граф ациклический. Пусть цикл существует. Рассмотрим рёбра, лежащие на этом цикле. Каждому ребру соответствует строка, в

которой осуществляется переход между вершинами. Каждой строке соответствуют две компоненты, между которыми проведено ребро. Выпишем сверху вниз номера компонент из самого левого столбца, входящих в цикл, — получим список a_i длины k . То же самое сделаем для самого правого столбца — получим список b_i длины k . Множества вершин в этих списках совпадают, так как они образуют цикл, и каждая вершина встречается там лишь единожды. Если для всех $1 \leq i \leq k$ выполняется $a_i = b_i$, то вершины цикл не образуют, так как каждое ребро — петля. Иначе существуют такие вершины x и y , что $pos_a(x) < pos_a(y)$ и $pos_b(x) > pos_b(y)$, где $pos_a(v)$ — позиция вершины v в списке a и $pos_b(v)$ — позиция вершины v в списке b . В таком случае компоненты, соответствующие вершинам x и y , обязаны пересечься хотя бы в одной клетке плитки, что противоречит тому, что это две разные компоненты связности.

Таким образом, доказали, что граф является DAG'ом с петлями (DAG — Directed Acyclic Graph — Ориентированный Ациклический Граф). В нём нужно узнавать, существует ли путь из одной вершины в другую, проходящий ровно по заданному количеству рёбер. Сначала рассмотрим случай, если в графе отсутствуют петли. Пусть $dp[x][z][k]$ — существует ли путь из x в z , проходящий ровно по k рёбрам. $dp[x][z][1] = 1$, если существует ребро из x в z . Для вычисления значений динамики при $k > 1$ переберём первое ребро на пути из x в z . Пусть оно ведёт в вершину y . Тогда $dp[x][z][k] = 1$, если нашёлся хотя бы один y , для которого $dp[y][z][k-1] = 1$. Заметим, что перебирать значения $k > h$ не имеет смысла, так как в графе не больше h вершин, и он ациклический. Пересчёт всех состояний динамики можно реализовать при помощи обхода в глубину. Это будет работать за $\mathcal{O}(h^3)$, так как для фиксированных z и k мы перебираем пары (x, y) , каждая из которых соответствует ребру в графе, которых не больше h .

Осталось разобраться, что делать с петлями и каким образом они влияют на ответ. На самом деле, они позволяют пройти часть пути из начальной вершины, потом отойти необходимое количество рёбер по петле и затем добраться до конечной вершины. Пусть $inf[x][z]$ — существует ли путь из x в z , проходящий хотя бы по одной вершине, имеющей петлю. Вычисление значений этой динамики можно осуществить при помощи того же обхода в глубину. Пусть путь из x в z существует, а d — минимальное, такое что $dp[x][z][d] = 1$. Тогда, если $inf[x][z] = 1$, то из x в z существуют пути всех длин, больших либо равных d . Таким образом, при посчитанных значениях dp и inf можно отвечать на запрос за $\mathcal{O}(1)$, и общее время работы программы будет $\mathcal{O}(h^3 + q)$.

Задача L. Задача из Катав-Ивановска

Автор задачи: Екатерина Шилыева. Разработчик задачи: Мария Потапова

Если $q \leq n$, то можно просто перебрать ходы и найти максимальный ответ на некотором префиксе ходов.

Иначе можно заметить, что за первые n игр барабаны вернутся в начальное состояние, соответственно образуется цикл из троек индексов ячеек на барабанах. Для $q > n$, если за первые n игр прибыль положительная, то выгодно брать наибольшее количество полных циклов, но надо разобрать 2 случая: берем максимальное количество циклов, которое можем пройти за q игр, и максимум из прибыли на префиксе остатка, либо на один цикл меньше с максимумом уже всего цикла.

Решение можно реализовать за $\mathcal{O}(n)$.

Задача M. Задача из Черёмушек

Автор и разработчик задачи: Александр Степаненко

При известных a , b , c мы сразу можем определить, какие площади должны получиться при разбиении. Для этого нужно найти исходную площадь многоугольника и разделить её в нужном отношении на три части. Заметим, что из формулы Пика следует, что площадь любого многоугольника с целочисленными координатами вершин на клетчатой бумаге либо целая, либо полуцелая. Поэтому можно удвоить все площади и работать только с целыми числами.

Теперь нужно найти количество возможных разбиений. Мысленно упорядочим площади слева направо при разрезании и скажем, что они должны быть S_a , S_b , S_c . В такой постановке первая диагональ отсекает часть площадью S_a , а вторая — S_c , при этом площадь между диагоналями будет S_b .

Обозначим первую диагональ как (i, j) , а вторую как (k, l) . Будем считать вершины упорядоченными в порядке обхода по часовой стрелке. Тогда многоугольник с вершинами $[i, i + 1, \dots, j - 1, j]$ (при этом j может быть меньше i , если мы перешли через n -ю вершину) будет иметь площадь S_a . Очевидно, что второй многоугольник не взаимодействует с вершинами первого, за исключением, возможно, крайних точек i, j .

Таким образом, при фиксированной диагонали (i, j) нам нужно подсчитать количество пар (k, l) таких, что они отрезают площадь S_c и лежат на оставшейся части многоугольника (исключая только вершины между i и j не включительно). Получаем тривиальное решение за $\mathcal{O}(n^4)$, при условии что мы можем быстро найти площадь, отсекаемую диагональю. Это легко сделать с помощью формулы Гаусса (формулы шнурования) и префиксных сумм.

Теперь заметим, что для каждого фиксированного i существует не более одного j такого, что площадь, отсекаемая диагональю (i, j) , будет равна S_a . Это, например, следует из монотонного увеличения площади по мере удаления вершины j при использовании формулы Гаусса. Аналогичное утверждение верно и для пар (k, l) с площадью S_c : для каждого k существует не более одного подходящего l . Поскольку у нас всего три требуемых значения площади, их все можно предподсчитать для каждой вершины с помощью бинарного поиска или метода двух указателей. Получаем решение за $\mathcal{O}(n^2)$.

Чтобы улучшить эту оценку, можно заметить, что при переборе пар (i, j) (где j вычисляется по i и требуемой площади за $\mathcal{O}(1)$) интервал для (k, l) сдвигается, и всего таких сдвигов не более n . Следовательно, с этим интервалом можно работать как со скользящим окном. Также можно наложить определённые ограничения на все подходящие пары (k, l) и находить нужное количество для данной диагонали (i, j) с помощью декартова дерева или ordered set. В итоге можно добиться сложности $\mathcal{O}(n \log n)$ или даже $\mathcal{O}(n)$. Стоит также отметить, что нужно аккуратно учитывать повторы при переборе.

Задача N. Задача из Коркино

Разработчик задачи: Мария Левина

Заметим, что строка отсортирована по неубыванию, если в ней нет подстроки **ba**.

Посчитаем префиксные суммы, где $pref_i$ — количество подстрок **ba** на префиксе. Посчитать такой массив очень просто: $pref_i = pref_{i-1}$, и если $s_{i-1}s_i = ba$, то нужно увеличить $pref_i$ на единицу. Тогда подстрока $s_ls_{l+1} \dots s_r$ отсортирована по неубыванию, если $pref_r - pref_l = 0$.

Решение можно реализовать за $\mathcal{O}(n + q)$.

Задача O. Задача из Кузнецкого

Автор и разработчик задачи: Александр Волков

Чтобы найти наименьший размер перемешанного отрезка, можно отсортировать копию исходного массива и найти первый и последний элементы, стоящие не на своих местах. Они и будут концами искомого отрезка. Максимальный по размеру отрезок — это всегда весь исходный массив.

Решение можно реализовать за $\mathcal{O}(n \log n)$.