,

# Разбор задачи «Хорошие массивы»

### $O(nc^2)$ **(29 баллов)**

Обозначим $dp[i][j]$ как количество массивов длины $i$, где на последней позиции стоит число со значением $j$. Изначально объявим все $dp[1][j] = 1$. Произвольное $dp[i][j]$ можно вычислить как $dp[i][j] = \sum_{j|d} dp[i-1][d]$ (перебираем $d$ от $j$ до $n$ и проверяем, делится ли $d$ на $j$). Состояний динамики $O(nc)$, каждое вычисляется за $O(c)$. Ответом будет $dp[n][1] + dp[n][2] + \ldots + dp[n][c]$

### $O(nc \log c)$ **(41 балл)**

В решении предыдущей группы можно заметить, что на $j$ делятся $j,\ 2j,\ 3j, \ldots$ и пересчитывать состояние $dp[i][j]$ как $dp[i-1][j] + dp[i-1][2j] + \ldots$ Тогда пересчет на слое происходит суммарно за $O(c \log c)$, слоев $O(n)$, поэтому общее время работы составит $O(nc \log c)$.

### $O(c \log c + n)$ **(71 балл)**

Обозначим количество хороших массивов, в которых первое число $x$, за $f(x)$. Ответом на задачу будет $f(1) + f(2) + \ldots + f(c)$. Научимся искать $f(p^k)$. Сначала будет идти $c_k$ чисел $p^k$, затем $c_{k-1}$ чисел $p^{k-1}$, ..., $c_0$ чисел $p^0$. $p^k$ должно встречаться хотя бы раз, остальные могут встречаться и 0 раз. Тогда $f(p^k)$ — количество рабзиений числа $n$ на 1 положительное и $k$ неотрицательных слагаемых, то есть $C^k_{n+k-1}$. $f(p_1^{\alpha_1} \ldots p_k^{\alpha_k}) = f(p_1^{\alpha_1}) f(p_2^{\alpha_2} \ldots p_k^{\alpha_k})$. Тогда массиву $a_1,\ a_2,\ \ldots,\ a_n$ будет соответствоть пара массивов $b_1,\ b_2,\ \ldots,\ b_n$ и $c_1,\ c_2,\ \ldots,\ c_n$, где $a_i = b_i c_i$ и $b_i$ - это $p_1$ в некоторой степени, $c_i$ - произведение $p_2,\ \ldots,\ p_k$ в некоторых степенях. Применив утверждение несколько раз, получим $f(p_1^{\alpha_1} \ldots p_k^{\alpha_k}) = f(p_1^{\alpha_1}) f(p_2^{\alpha_2}) \ldots f(p_k^{\alpha_k})$. С помощью решета Эратосфена найдем для каждого числа минимальный делитель. Тогда для нахождения $f(x)$ необходимо разложить $x$ на множители за $O(\log c)$, посчитать для них ответы и перемножить. Чтобы искать $C^k_n$, можно преподсчитать факториалы и обратные к ним. Общее время работы - $O(c \log c)$.

### $O(c + n)$ **(86 баллов)**

Насчитать все $f(i)$ можно с помощью решета Эратосфена за $O(c)$. Для каждого числа $i$ считаем его наименьший делитель $p$ и степень его вхождения в разложение $d$. Тогда $f(i) = f(p^d) \cdot f(\frac{i}{p^d}) = C^d_{n+d-1} \cdot f(\frac{i}{p^d})$.

### $O(c)$ **(100 баллов)**

Можно не считать все факториалы и обратные к ним. Достаточно заранее посчитать все $C^d_{n+d-1}$ (каждое за $O(d)$) так как $d$ может принимать не более $O(\log C)$ различных значений.

# Разбор задачи «Два проспекта»

Сначала поймем как для двух ребер $e_1$, $e_2$ найти количество тугриков. Уберем оба ребра $e_1$, $e_2$. Каждая из пар, вершины которой в разных компонентах связности, заплатит хотя бы один тугрик. Если вершины какой-то пары оказываются в разных компонентах связности и при удалении только ребра $e_1$, и при удалении только ребра $e_2$, то такая пара заплатит два тугрика в любом случае. Таким образом, такие пары заплатят ровно два тугрика, а те пары, вершины которых только при удалении обоих ребер попадают в разные компоненты связности, заплатят ровно один тугрик. Количество этих пар каждого вида можно вычислить за $O(m+k)$, сделав поиск компонент связности с помощью dfs в каждом из случаев.

Этот алгоритм подсчета количества тугриков для заданной пары ребер дает нам решение первой подгруппы за $O(m^2(m+k))$ — давайте просто переберем все пары ребер.

Заметим, что граф при удалении обоих ребер должен стать несвязным, иначе ответ для этой пары ребер 0. Значит хотя бы одно ребро должно попасть на любое остовное дерево, например дерево dfs. Значит мы можем перебрать $O(nm)$ кандидатов пар ребер (одно из ребер перебираем на дереве dfs), что дает нам решение второй подгруппы за $O(nm(m+k))$.

Зафиксируем ребро дерева dfs, которое будет в паре. Удалим его. Заметим, что другое ребро, которое мы выберем в пару, должно быть мостом оставшегося графа (а иначе вообще не важно что это за ребро). Найдем все мосты этого графа. Для каждого моста можно легко посчитать ответ при его выборе суммарно за $O(m + k)$ для всех мостов. Это дает нам решение третьей подгруппы за $O(n(m + k))$.

Пусть ответ это пара ребер $e_1$, $e_2$, где $e_1$ лежит на дереве dfs. Давайте поймем, какими могут быть ребра $e_1$ и $e_2$. Если $e_1$ это мост, то $e_2$ тоже должно быть мостом (иначе при удалении $e_1$ ребро $e_2$ не будет мостом). Если $e_1$ это не мост, то каким может быть ребро $e_2$? Мы знаем, что $e_2$ должно быть мостом в графе без ребра $e_1$. Если $e_2$ не лежит на дереве dfs, то это должно быть единственное ребро, которое накрывает $e_1$, так как иначе $e_2$ не будет мостом графа без ребра $e_1$. Если $e_2$ лежит в дереве dfs, то множества внешних ребер, накрывающих $e_1$ и накрывающих $e_2$ должны совпадать. На самом деле, случай того, что $e_2$ это не мост графа без $e_1$ возможен, но тогда неважно какое ребро $e_2$ взять и количество тугриков будет равно количеству пар, вершины которых будут в разных компонентах связности после удаления $e_1$. Поэтому $e_1$ должно быть мостом.

Таким образом есть следующие случаи того, какими могут быть ребра $e_1$, $e_2$:

1. Оба ребра $e_1$, $e_2$ являются мостами графа.
2. Ребро $e_1$ является мостом графа, ребро $e_2$ не важно.
3. Ребро $e_1$ лежит на дереве dfs, а ребро $e_2$ это единственное внешнее ребро, которое его накрывает.
4. Ребра $e_1$, $e_2$ лежат на дереве dfs, для них множества внешних ребер, которые их накрывают совпадают.

Для каждого случая научимся находить пару ребер $e_1$, $e_2$, которая дает максимальный ответ и затем выберем среди них максимальную.

Сопоставим каждой паре вершин путь в дереве dfs между ее вершинами.

В случае 1 заметим, что для пары мостов $e_1$, $e_2$ ответ равен сумме количества путей, содержащих ребро $e_1$ и количества путей, содержащих ребро $e_2$. Давайте тогда для каждого ребра $e$ дерева dfs посчитаем $c_e$ — количество путей, содержащих $e$. Это можно сделать прибавив на каждом из $k$ путей 1, что можно сделать с помощью LCA и прибавления $\pm 1$ в концах вертикальных путей и префиксных сумм. Далее надо выбрать два моста $e_1$, $e_2$, таких что $c_{e_1} + c_{e_2}$ максимально, для этого надо просто найти два моста с максимальным значением $c_e$.

Заметим, что в случае 2 для моста $e_1$ ответ равен $c_{e_1}$. Тогда в этом случае оптимально взять мост с максимальным значением $c_e$. В качестве $e_2$ можно взять любое ребро.

В пятой подгруппе граф является деревом, поэтому все его ребра это мосты. Поэтому возможны только случаи 1, 2, которые мы уже научились решать за $O((n + k) \log n)$.

Для каждого ребра $e$ дерева dfs посчитаем $f_e$, $h_e$ — количество внешних ребер графа, накрывающих $e$, хеш множества внешних ребер графа, накрывающих $e$. Для того, чтобы посчитать хеш, можно каждому внешнему ребру сопоставить случайное 64-битное число, а хеш будет равен сумме этих чисел по всем накрывающим ребрам. Эта часть делается за $O(n)$.

Тогда в случае 3 нам интересны ребра $e_1$, для которых $f_{e_1} = 1$. Ответ для пары ребер $e_1$, $e_2$ тогда будет равен $c_{e_1}$. Переберем тогда все ребра дерева dfs, проверим критерий и найдем максимальный ответ.

Остался случай 4. Этот случай гораздо более сложный, чем случаи 1, 2, 3. В этом случае нам интересны пары ребер $e_1$ $e_2$, такие что $h_{e_1} = h_{e_2}$. Заметим, что если $h_{e_1} = h_{e_2}$, то ребро $e_1$ лежит на вертикальном пути от корня дерева dfs до $e_2$ (Н.У.О. ребро $e_1$ выше). Значит все ребра с одинаковым хешом лежат на некотором вертикальном пути. Тогда ответ для пары ребер $e_1$, $e_2$ будет равен количеству путей, которые накрывают ровно одно ребро среди $e_1$ и $e_2$. Заметим, что мы можем разделить каждый путь на два вертикальных и от этого ничего не поменяется. Поэтому мы можем считать, что все пути вертикальные. Началом пути будем называть более высокий конец, концом более низкий.

,

Будем делать обход в глубину дерева dfs и поддерживать дерево отрезков с операциями прибавления на отрезке и максимума на отрезке. При этом его ячейки будут соответствовать ребрам на пути от корня до текущего ребра $e_2$. В ячейке, соответствующей ребру $e_1$ будет записан ответ для пары ребер $e_1$, $e_2$, если $h_{e_1} = h_{e_2}$. Поймем, как пересчитывать это ДО и как бороться с парами ребер $h_{e_1} \neq h_{e_2}$.

Когда мы стоим в вершине $v$, последнее ребро $e$ и мы переходим по ребру $s$ нам для каждого ребра $e_1$ надо в ячейке для $e_1$ поменять число с ответа для $(e_1, e)$ на ответ для $(e_1, s)$. Какие пути могли накрывать/не накрывать ровно одно из $e_1$, $e$, но стать не накрывать/накрывать ровно одно из $e_1$, $s$?

1. Путь, начало которого это $v$ и который накрывает ребро $s$, прибавляет 1 во все ячейки.

2. Путь, конец которого это $v$ или конец которого лежит в поддереве $v$, но не в том куда идет ребро $s$, вычитает 1 для всех ребер $e_1$, которых этот путь не накрывал и прибавляет 1 для всех ребер $e_1$, которых этот путь накрывал. Назовем такие пути сложными.

Случай 1 простой: можно просто перебрать такие пути и сделать прибавление 1 на всем ДО.

Случай 2 намного сложнее. Количество сложных путей во всех $v$ может быть большим и суммарно быть $O(nk)$. Заметим, что в случае, когда наш граф это цикл, количество сложных путей во всех $v$ будет суммарно $O(k)$, это просто пути, которые кончаются в $v$. Поэтому такой же перебор как в случае 1 сработает. Поскольку у всех ребер хеши одинаковые в этом случае, это дает нам решение шестой подзадачи за $O((n+k)\log n)$.

Поймем, что делать с условием равенства хешей. Назовем кластером ребра дерева dfs с одинаковым хешом. Сопоставим каждому кластеру вертикальный путь от первого ребра кластера до последнего. Заметим, что на каждом таком пути внутри могут быть ребра с другим хешом. Заметим тогда, что либо пути для двух кластеров совсем не пересекаются (по ребрам), либо один из путей вложен в другой и при этом располагается между соседними вхождениями хеша для большего пути.

Тогда когда мы переходим по ребру $e_2$ в ДО будем искать минимум на отрезке от самого высокого ребра, с таким же хешом, что у ребра $e_2$. Но что делать с ребрами между ними, которые имеют другой хеш? Заметим, что делая такой обход, мы можем прибавить $-\infty$ на отрезке от последнего ребра с таким же хешом, что у $e_2$, до ребра $e_2$. Тогда мы перестанем рассматривать эти ребра в ответ. И при этом никакие из хешей этих ребер никогда больше не встретятся из-за устройства кластеров, поэтому мы дальше ничего не испортим.

Надо понять, как обновлять с помощью сложных путей из случая 2. Посмотрим на величину $c_e - c_s$. Это в точности количество сложных путей минус количество путей, которые начинаются в $v$ и содержат $s$. Последнее количество мы знаем, тогда мы находим $x$ — количество сложных путей. Добавим $x$ ко всему ДО. Теперь нам нужно вычесть 2 на некоторых префиксах, заканчивающихся в началах сложных путей. Заметим, что мы можем этого не делать для сложного пути, если хеши с соответствующего префикса больше не встретятся ниже в обходе. Заметим, что каждый путь $P$ будет сложным и требующим вычитания на префиксе ровно в одной вершине $v$. Это так, потому что если выделить пути кластеров, которые ни в кого не вложены, то это будут не пересекающиеся вертикальные пути. И то, в какой внешний путь кластеров попало начало пути $P$, определяет единственную вершину $v$, в которой этот путь будет сложным и потребует прибавления на префиксе: эта вершина $v$ это LCA конца внешнего пути кластера и конца пути $P$. Можно отдельным dfs для каждой вершины $v$ найти какие сложные пути потребуют прибавления на префиксе в ней. Суммарное количество таких путей будет $O(k)$. В итоге мы разобрали случай 2 с помощью $O(n+k)$ операций с ДО.

В итоге получается полное решение за $O(m + (n+k)\log n)$.

# Разбор задачи «Странная сумма»

Заметим, что манхеттенское расстояние между клетками $(r_1, c_1)$ и $(r_2, c_2)$ — равно $|r_1 - r_2| + |c_1 - c_2|$. В качестве одного из кратчайших путей мы можем рассмотреть путь, который в начале меняет первую координату, а потом — вторую.

Чтобы решить первую подзадачу, достаточно перебрать все пары клеток, и если две клетки в паре имеют одинаковый цвет, добавить расстояние между ними к ответу.

Решим четвёртую подзадачу, где количество цветов равно двум. Заметим, что кратчайших путь между двумя клетками пересекает границу между строками $r$ и $r+1$ когда первая координата одной из клеток не больше $r$, а другой — не меньше $r+1$. Для каждого префикса и суффикса строк посчитаем общее количество клеток каждого из цветов, после этого мы можем за $O(1)$ вычислить количество путей, пересекающих каждую границу между строками. Аналогично посчитаем количество путей, пересекающих границы между столбцами. Поскольку длина пути — это количество раз, которое он пересекает границы, мы можем просуммировать все количества пересечений границ, и получить ответ.

Решим вторую подзадачу, где есть всего одна строка. С помощью хеш-таблицы мы можем для каждого цвета получить список позиций клеток с этим цветом. Пусть эти позиции $c_0, c_1, \ldots, c_{k-1}$. Тогда мы хотим посчитать $\sum_{i=0}^{k-1}\sum_{j=i+1}^{k-1}|c_i - c_j|$. Отсортируем массив $c$ и назовём результат $s$. Тогда:

$$\sum_{i=0}^{k-1}\sum_{j=i+1}^{k-1}|c_i - c_j| = \sum_{i=0}^{k-1}\sum_{j=i+1}^{k-1} s_j - s_i = \left(\sum_{i=0}^{k-1}\sum_{j=i+1}^{k-1} s_j\right) + \left(\sum_{i=0}^{k-1}\sum_{j=i+1}^{k-1} -s_i\right)$$

Значение $s_j$ встречается в первой двойной сумме в точности $j$ раз, значение $-s_i$ встречается во второй сумме в точности $k-1-j$ раз. Тогда, сумма равна:

$$\sum_{j=0}^{k-1} j s_j + \sum_{i=0}^{k-1} -(k-1-i)s_i = \sum_{i=0}^{k-1}(2i+1-k)s_i$$

Последнюю сумму можно вычислить за линейное время.

Чтобы решить третью подзадачу, достаточно использовать решение второй подзадачи, но нужно для каждого цвета добавить к ответу произведение количеств его вхождений в первую и вторую строку.

Для полного решение нужно было заметить, что задачу можно решать отдельно по строкам и столбцам. Переберём цвет, пусть множество клеток определённого цвета — $(r_0, c_0), \ldots, (r_{k-1}, c_{k-1})$. Искомая сумма равна:

$$\sum_{i=0}^{k-1}\sum_{j=i+1}^{k-1}|r_i - r_j| + |c_i - c_j| = \left(\sum_{i=0}^{k-1}\sum_{j=i+1}^{k-1}|r_i - r_j|\right) + \left(\sum_{i=0}^{k-1}\sum_{j=i+1}^{k-1}|c_i - c_j|\right)$$

Мы можем посчитать первую и вторую сумму аналогично второй подзадаче. Асимптотика решения — $O(nm\log(nm))$. Мы можем не сортировать массивы, а добавлять клетки в хеш-таблицу в правильном порядке, тогда получим асимптотику $O(nm)$.

# Разбор задачи «Бизнес-шоу»

### $q = 1$ (10 баллов)

Мы не можем не активировать единственное возможное предложение. Теперь известно, какие клетки включены, а какие нет. Так что мы можем посчитать ответ с помощью стандартной динамики (путь максимальной стоимости в прямоугольнике).

### Решение за $O(n^2q)$ (16 баллов)

Пусть $pref[i][j]$ — сумма первых $j$ элементов в $i$м ряду. Тогда определим $s[i] := pref[0][i+1] - pref[1][i]$, $f[i] := pref[1][i+1] - pref[2][i] + pref[2][n]$.

Заметим, что если мы вошли во второй ряд в клетке $(2, i)$ и вышли из него в клетке $(2, j)$, то суммарное изменение баланса от прохождения по клеткам равно $s[i] + f[j]$.

,

Теперь заметим, что теперь задача свелась к тому, чтобы максимизировать $s[i] + f[j] - cost(i, j), i \leqslant j$, где $cost(i, j)$ — минимальная стоимость разблокировки отрезка $[i, j]$.

Теперь посчитаем $dp[i]$ — максимальная выгода, которую можно получить, дойдя до клетки $(2, i)$, учитывая стоимости отрезков, которые мы разблокировали.

Пересчет: переберем отрезок, покрывающий $i$, и прорелаксируем $dp[i]$ значением $\max\limits_{l-1 \leqslant j < i} dp[j] - k$. Также мы можем войти во второй ряд в точке $i$, тогда прорелаксируем через $\max\limits_{l \leqslant j < i} s[j] - k$.

Ответом будет $\max\limits_{0 \leqslant i < n} f[i] + dp[i]$

### Решение за $O(n\,q\,log\,n)$ (+14 баллов)

Заметим, что для пересчета динамики из предыдущего решения можно воспользоваться деревом отрезков, и тогда вся динамика будет пересчитываться за $O(n\,q\,log\,n)$

### Все $k_i$ одинаковы (+17 баллов)

Будем оптимизировать пересчет динамики. Заметим, что для предложения $l, r, k$ оно будет корректно для пересчета с момента, когда пересчитываем $dp[l]$ до момента, когда пересчитываем $dp[r]$ включительно.

Теперь у нас есть запросы вида «Добавить/удалить отрезок для пересчета динамики» и «Посчитать $dp[i]$».

Заметим, что добавить отрезок, это то же самое, что сказать, что можно использовать все $dp[j]$ на отрезке для перехода стоимости $k$.

То есть, задача теперь свелась к запросам «Включить/выключить отрезок динамики» и посчитать минимум по включенным значениям на отрезке. Это все можно реализовать с помощью дерева отрезков за $O(log\,n)$ на запрос

### Решение за $O(n\,log^2 n)$ (+21 балл)

Обобщим подход для всех одинаковых $k_i$.

Заметим, что чтобы пересчитать динамику, достаточно для каждого $j$ пересчитываться только через самый дешевый отрезок. Теперь запросы имеют вид «Добавить/удалить возможность перехода из $dp[j]$ на отрезке за $k$ рублей» и «Посчитать максимум по $dp[j]$ − (минимальная стоимость перехода из $j$ в $i$)».

Чтобы отвечать на эти запросы достаточно хранить в каждой вершине дерева отрезков `set` из стоимостей предложений, в который входит соответствующий этой вершине отрезок, и при пересчете ответа для вершины брать из этого `set` минимальный элемент и максимальное $dp$ на этом отрезке.

### Полное решение.

Посчитаем $dp[i]$ — аналогичная динамика, но известно, что самый правый отрезок, который мы взяли, заканчивается в $i$.

Теперь переходов суммарно $O(q)$, посчитаем эту динамику за $O(q\,log\,n)$.

Рассмотрим множество отрезков, которое является оптимумом (при его включении достигается максимальная выгода).

Переберем отрезок, который может быть самым правым в этом оптимуме. Заметим, что максимальная выгода, если этот отрезок самый правый, и есть хотя бы один другой отрезок, который мы взяли, будет равен $\max\limits_{l \leqslant i \leqslant j \leqslant r} dp[i] + f[j] - k$ (предпоследний отрезок закончился в $i$, вышли из $j$). Для каждого отрезка эту величину можно посчитать с помощью дерева отрезков за $O(log\,n)$.

А если берем только один отрезок, то надо прорелаксировать ответ $\max\limits_{l \leqslant i \leqslant j \leqslant r} s[i] + f[j] - k$.

Итого имеем решение за $O(q\,log\,n)$.

# Разбор задачи «Растягивание плоскости»

Группа 1

Реализуем наивное решение: для каждого запроса переберём все пары точек за $O(n^2 \cdot q)$.

Группы $2-3$

Заметим, что пара точек, расстояние между которыми наибольшее всегда лежит на выпуклой оболочке исходного множества точек, так как множество точек лежащих на выпуклой оболочке не меняется от запроса к запросу, построить выпуклую оболочку можно 1 раз в целых числах за $O(n^2)$.

Затем, для ответа на запрос воспользуемся одним из стандартных алгоритмов, позволяющих найти 2 наиболее удалённые точки выпуклого многоугольника. Это можно сделать за $O(n)$ с помощью вращающихся калиперов, или за $O(n \cdot log(n))$ найдя для каждой стороны не более двух наиболее удалённых от неё точек, с помощью поиска касательной к выпуклому многоугольнику параллельной данной прямой, и прорелаксировав ответ через пары, состоящие из точек на стороне и этих $1-2$ точек.

Группы $4-5$

Так как все точки случайные, размер выпуклой оболочки $O(\log(n))$, в 4-й группе можно реализовать наивное решение из 1-й группы, дополнительно построив выпуклую оболочку, в 5-й достаточно реализовать решение, описанное в $2-3$ группах построив выпуклую оболочку за $O(n \cdot \log(n))$.

Обсудим идею, приближающую нас к полному решению:

Назовём выпуклую оболочку исходного множества точек $A$. Тогда утверждается, что расстояние между двумя наиболее удалёнными точками в этом множестве, это максимальное расстояние от начала координат до точки лежащей в множестве $B$ — сумме Минковского $A$ и $-A$. Заметим, что $B$ при растяжении $A$ в $\alpha$ раз так же растягивается в $\alpha$ раз. Таким образом задача сводится к следующей: дано $O(n)$ пар точек $(x, y)$, требуется отвечать на запрос $\max_{(x,y)\in B} \sqrt{x^2 \cdot \alpha^2 + y^2} = \sqrt{\max_{(x,y)\in B} (x^2 \cdot \alpha^2 + y^2)}$, таким образом мы сводим задачу к поиску максимума в точке среди множества линейных функций, которая решается с помощью convex hull trick за $O(q \cdot \log(n))$.

Группы $6-7$

В этих группах нужно было выделить множество линейных функций за квадрат, воспользовавшись тем фактом, что для каждой стороны $1-2$ наиболее удалённые от неё точки не изменяются, этот доказывается через то, что от растяжения плоскости в $\alpha$ раз, знаки векторных произведений векторов на этой плоскости не меняются.

Затем, построим на этом множестве линейных функцих выпуклую оболочку и будем отвечать на запросы с помощью бинарного поиска.

Таким образом получается решение за $O(n^2 + (n+q) \cdot \log(n))$.

Группы $8-9$

Для полного решения нужно было выделить множество линейных за окололинейное время, а, затем воспользоваться решением для групп $6-7$.

Итого: $O((n+q) \cdot \log(n+q))$, 100 баллов и никаких операций с вещественными числами, помимо поиска оптимума среди множества линейных функций.

# Разбор задачи «Целостный массив»

1. Для решения первой группы достаточно реализовать наивный алгоритм за $O(N^3)$. Для этого переберём каждые два числа массива $a$ и дополнительным проходом по всему массиву проверим, что результат деления тоже лежит в массиве $a$.

2. Для решения второй группы можно было заметить, что $C \leqslant 10\,000$. Тогда можно реализовать следующее решение за $O(C^2)$: построим массив $cnt$ размера $c$, где $cnt_x$ означает количество вхождений числа $x$ в массив $a$. Теперь переберём все возможные пары чисел $x, y$ от 1 до $c$, которые есть в массиве $a$ (для них $cnt \neq 0$), и проверим, что результат деления тоже есть в $a$, то есть $cnt_{res} \neq 0$. Поскольку такая проверка выполняется за $O(1)$, то весь алгоритм работает за $O(C^2)$ — перебор всех возможных пар чисел.

,

3. Для решения третьей группы достаточно оптимизировать решение первой группы. Мы точно так же будем перебирать все возможные пары чисел в массиве $a$ за $O(N^2)$, но проверку того, что результат деления есть в массиве, теперь будем делать за $O(1)$. Для этого достаточно воспользоваться хеш-таблицей.

4. Для решения четвёртой группы необходимо придумать более оптимальное решение. Для начала научимся следующей примитиве: проверить, есть ли в массиве число от $l$ до $r$ за $O(1)$. Для этого построим массив $cnt$, как и раньше, для которого $cnt_x$ — количество чисел $x$ в массиве $a$. Теперь построим на этом массиве префиксные суммы $pref$. Теперь в массиве есть число от $l$ до $r$ тогда и только тогда, когда $pref_r - pref_{l-1} > 0$, что проверяется за $O(1)$.

   Теперь перейдём к самому решению за $O(N\sqrt{C})$. Пусть $x$ есть в массиве $a$. При фиксированном результате деления $\left\lfloor \frac{x}{d} \right\rfloor = res$, знаменатель $d$ пробегает некоторый отрезок значений от $l$ до $r$ (для любого $d$ от $l$ до $r$ выполнено равенство $\left\lfloor \frac{x}{d} \right\rfloor = res$). Тогда если в массиве $a$ есть число $l \leqslant d \leqslant r$, то и $res$ должно быть в $a$.

   Пусть мы уже знаем $l$, тогда $res = \left\lfloor \frac{x}{l} \right\rfloor$, $r = \left\lfloor \frac{x}{res} \right\rfloor$. Тогда просто переберём $l$ от 1 до $x$ и посчитаем $res$ и $r$. С помощью примитивы проверим, есть ли в $a$ хотя бы одно число от $l$ до $r$ (это и будет $d$). Если так, то $res$ тоже должен быть в $a$. После проверки перейдём к следующей левой границе: $l = r + 1$. Осталось заметить, что такой алгоритм работает за количество различных значений $res$ (за одну итерацию мы гарантированно рассматриваем новое значение $res$), а их $O(\sqrt{C})$. Для каждого $x$ из $a$ мы проделываем все эти действия, тем самым получая решение за $O(N\sqrt{C})$.

5. Для пятой группы приведём решение, работающее за $O(C \log C)$. Воспользуемся тем, что $\left\lfloor \frac{x}{y} \right\rfloor = r$ тогда и только тогда, когда $x$ принимает значение от $y \cdot r$ до $y \cdot (r + 1) - 1$ включительно. Зафиксируем такие $y$ и $r$, что $y$ лежит в $a$, а число $r$ — нет. Тогда если есть число $x$ из массива $a$, лежащее в указанном отрезке, то массив не является целостным (потому что $\left\lfloor \frac{x}{y} \right\rfloor = r$, $r \notin a$). Значит, нам просто нужно проверить, что в указанном отрезке нет ни одного числа, а это мы уже умеем из примитивы. Мы научились проверять фиксированную пару $r$, $y$ за $O(1)$, тогда переберём все возможные пары в порядке возрастания сначала по $r$, которое не лежит в $a$, потом по $y$, лежащему в $a$. Заметим, что от текущего $r$ можно перейти к $r + 1$ как только $r \cdot y > c$. Действительно, это означает, что левая граница указанного отрезка больше $c$, то есть интересующего нас $x$ точно нет, и ничего проверять не надо. За счёт этой оптимизации решение и работает за $O(C \log C)$.

# Разбор задачи «Задача для третьеклассника»

Пусть $K$ - размер алфавита, то есть номер максимальной буквы, которая в нем встречается.

Для начала посчитаем, сколько вообще различных строк можно составить, если у нас в распоряжении $c_1$ букв 1-го типа, $c_2$ букв 2-го типа, ..., $c_K$ букв типа $K$. Это школьная формула:

$P(c_1, c_2, \ldots, c_K) = \frac{(c_1+c_2+\ldots+c_K)!}{c_1! \cdot c_2! \cdot \ldots \cdot c_K!}$

чтобы быстро считать её для разных $c_1, c_2, \ldots, c_k$ заранее предподсчитаем все факториалы и обратные к ним числа по модулю $C = 998244353$ за $\mathrm{O}(n \cdot \log C)$

Чтобы строка $x$ была меньше строки t, нужно чтобы у них совпадал некоторый префикс. Переберем длину этого совпадающего префикса от 0 до $\min(n, m)$. Если у строк $x$ и $t$ совпадают первые $i$ символов, то мы точно знаем, сколько каких букв осталось у нас в распоряжении. Чтобы это поддерживать, заведем массив cnt, на $i$-й позиции которого будет количество оставшихся букв типа $i$.

Переберём букву, которая будет стоять сразу после совпадающего префикса. Чтобы полученная строка была меньше $t$, эта буква должна быть строго меньше соответствующей буквы в $t$, а все последующие буквы можно расставить в любом порядке. Посчитаем количество рассмотренных таким образом строк $x$ по формуле выше.

Единственный случай, когда полученнная строка $x$ может оказаться лексикографически меньше $t$, который мы не посчитаем - это тот, где она является префиксом строки $t$, но имеет меньшую длину. Отдельно проверим, можем ли мы получить такую строку, и если да - добавим к ответу 1.

,

Так как на каждом, из не более $\min(n, m)$ шагов нам нужно перебирать не более $K$ вариантов следующей буквы, и каждый вариант мы просчитываем за $O(K)$ - получаем асимптотику $O(\min(n, m) \cdot K^2 + n \cdot \log C)$

Чтобы ускорить полученное решение, заведем массив $add$, в $i$-й ячейке которого сохраним, сколькими способами можно будет расставить оставшиеся буквы, если на текущюю позицию поставить букву $i$. Фактически $add_i = \frac{(cnt_1+cnt_2+...+cnt_K-1)!}{cnt_1!\cdot cnt_2!\cdot....\cdot(cnt_i-1)!\cdot....\cdot cnt_K!}$

Если мы научимся поддерживать этот массив, то на каждом шаге нам всего лишь нужно будет взять сумму элементов на некотором его префиксе. Посмотрим, как он изменяется, если очередная буква в строке $t$ это $i$, то есть $cnt_i$ должно уменьшиться на 1.

Для всех ячеек $j \neq i$ $add_j$ заменяется на $add_j \cdot \frac{cnt_i}{cnt_1+cnt_2+...+cnt_K-1}$. Чтобы применять модификации ко всему массиву, заведём отдельную переменную $modify$, на которую нужно домножить значение в ячейке, что получить значение, которое должно там лежать.

Для ячейки $i$ $add_i$ заменится на $add_i \cdot \frac{cnt_i-1}{cnt_1+cnt_2+...+cnt_K-1}$. А с учетом того, что мы ко всем ячейкам применили модификатор, значение $add_i$ достаточно домножить на $\frac{cnt_i-1}{cnt_i}$

С такой оптимизацией мы на каждом шаге теперь тратим только O(K) действий, чтобы посчитать префиксную сумму, и $O(\log(C))$ чтобы посчитать, на что нужно домножать ячейки массива. Получаем асимптотику $O(\min(n, m) \cdot (K + \log(C)))$

Чтобы избавиться от $K$ в асимптотике заметим, что единственное, что мы хотим делать с массивом $add$ - это брать сумму на префиксе и изменять значение в точке. Это можно реализовать за $O(log(K)$ с помощью Дерева Фенвика или Дерева Отрезков. Применив их получаем итоговую асимптотику $O(\min(n, m) \cdot (\log(K) + \log(C)))$.

На самом деле от $\log(C)$ в асимптотике можно избавиться, предподсчитав обратные по модулю для всех чисел от 1 до $n$ быстрее, чем за $O(n \cdot \log(C))$, но в данной задаче это не требовалось.

# Разбор задачи «Авиареформа»

Опишем формально условия задачи. Нам дан взвешенный граф, взвешенный граф на $n$ вершинах и $m$ рёбрах. Для него строится граф дополнение, в котором вес ребра равен минимальному максимуму на пути между концами ребра по исходному графу. По такому же принципу считаются веса рёбер исходного графа относительно графа дополнения. Требуется вывести итоговые веса рёбер исходного графа.

Для группы где $n \leqslant 100$ подходило следующее решение: Сделаем аналог алгоритма флойда, но вместо кратчайшего пути будем искать путь с минимальным максимумом. Так за $O(n^3)$ можно будет найти веса всех рёбер в графе дополнении. Запустившись второй раз, можно найти веса рёбер исходного графа.

Теперь заметим основную идею, которая пригодится для всех оставшихся групп. Заметим, что путь с минимальным максимумом проходит по минимальному остовному дереву. Это означает, что в графе дополнении достаточно построить минимальное остовное дерево, а дальше по нему веса исходных рёбер будет восстанавливаться за счёт запроса максимума на пути (что делается за $O(m \log n)$).

Для группы где $N \leqslant 10000$ и $n \leqslant 1000$ построим остовное дерево в исходном графе за $O(n^2 \log n)$. С помощью него найдём итоговые веса рёбер в графе дополнении. На них построим остовное дерево в графе-дополнении за $O(n^2 \log n)$. За $O(m \log n)$ найдём итоговые веса рёбер в исходном графе.

Для группы когда веса не больше 2 построим компоненты связности по рёбрам веса 1 в исходном графе. Возьмём любую вершину оттуда, и найдём все достижимые вершины компоненты по рёбрам графа дополнения. Заметим, что только такие вершины в графе дополнении могут быть соединены рёбрами веса 1, между остальными будут рёбра веса 2. Тогда будем действовать следующим образом: выберем любую компоненту связности по рёбрам из 1. Далее рассмотрим только вершины и рёбра, которые в ней есть. Нам требуется найти в дополнении такого графа компоненты связности. Для этого выберем некоторую вершину $A$. Будем хранить текущее множество вершин, которые достижимы от $A$ в графе дополнении (пусть это $S$). Добавим в очередь все рассматриваемые вершины. Будем по одной доставать вершины из очереди, и если между ними и $A$ нет ребра, то будем добавлять в компоненту связности дополнения графа по вершине $A$, иначе будем заново класть в очередь. Когда вершина $B$ в следующий раз достаётся из очереди, то проверим, что $S$ увеличилось

с момента прошлого доставания (если не увеличилось, то новые вершины точно не добавятся). Если же $S$ увеличилось, то мы знаем, что от всех старых вершин в $S$ не было рёбер до $B$ в графе дополнении, но они могли бы появиться от вновь добавленных вершин. Тогда переберём их и проверим, есть ли от них рёбра в $B$. Когда нашли компоненту связности, то переходим к новой вершине и повторяем тоже самое. Заметим, что на каждое неудачное доставание вершины из очереди должно быть ребро между ней и какой-то вершиной в компоненте. Значит всего неудачных доставаний из очереди $O(m)$, а значит общее время работы этого алгоритма $O(m)$. После нахождения компонент связности, соединим все вершины в них произвольным остовным деревом из единичных рёбер, остальные вершины соединим рёбрами веса 2.

Для полного решения надо использовать похожие идеи. Как в алгоритме Крускала будем строить минимальный остов исходного графа. В процессе построения будут храниться компоненты связности в исходном грфае по добваленным рёбрам. Внутри них будем хранить компоненты связности графа дополнения.

При добавлении нового ребра если объединились две компоненты связности исходного графа, то посмотрим, как изменятся компоненты связности графа дополнения. Заметим, что при объединении компонент связности исходного графа $S$ и $T$, если в них есть компоненты связности графа дополнения $A$ и $B$, то они не могут объединится только если между каждой вершиной из $A$ и из $B$ есть рёбра исходного графа. Тогда переберём все такие пары вершин и проверим, есть ли между ними рёбра. Заметим, что при таком объединении между этими компонентами добавится ребро в графе дополнении с весом, равным весу текущего добавленного ребра Крускалом. Тогда храня компоненты связности в графе дополнении и объединяя их, мы сможем получить остовное дерево графа дополнения. При этом в процессе работы на каждую неудачную попытку объединание компонент будет существовать ребро исходного графа между ними, значит общее число неудачных попыток объединения будет не больше $O(m)$. Получаем, что так за $O(m \log m)$ мы сможем построить остовное дерево в графе дополнении.