Всероссийская олимпиада школьников по информатике 2022–2023

Региональный этап

Разбор задач

Условия задач, тесты, решения и разбор задач подготовили Александр Бабин, Константин Бац, Никита Голиков, Владимир Новиков, Даниил Орешников, Михаил Первеев, Андрей Станкевич, Григорий Хлытин.

Ценные замечания по результатам тестирования задач сделали Антон Белый, Даниил Васильев, Дмитрий Дубровин, Мария Жогова, Михаил Первеев, Владимир Рябчун, Алексей Упирвицкий.

Задача 1. Разделение прямоугольника

Автор задачи: Андрей Станкевич

Пусть было проведено x горизонтальных и y вертикальных разрезов ($0 \leqslant x < a, 0 \leqslant y < b$).

Подзадача 1

Так как по условию подзадачи $a = 1$, то единственно возможное количество горизонтальных разрезов $x = 0$. Тогда количество вертикальных разрезов $y = k$. Остается проверить: если выполняются условия $k < b$ и $m = k + 1$, то можно провести $x = 0$ горизонтальных и $y = k$ вертикальных разрезов так, чтобы получилось ровно $m = k + 1$ частей, в противном случае поле нельзя разрезать требуемым образом.

Асимптотика такого решения $O(1)$.

Подзадачи 2 и 3

Переберем возможное количество горизонтальных разрезов x ($0 \leqslant x \leqslant k$), по возрастанию. Вычислим количество вертикальных разрезов $y = k - x$. Заметим, что если мы провели ровно x горизонтальных и ровно y вертикальных разрезов, то поле разделилось на $p = (x+1) \cdot (y+1)$ частей. Остается проверить: если $p = m$, то поле можно разрезать требуемым образом и мы нашли ответ $\{x, y\}$. Если для всех возможных x верно что $p \neq m$, то поле нельзя разрезать требуемым образом.

Асимптотика такого решения $O(k)$.

Подзадача 4

Аналогично предыдущей подзадаче хотим перебирать возможное количество горизонтальных разрезов x ($0 \leqslant x \leqslant k$). Будем делать это с учетом того, что число m должно делится на число $x+1$. Для этого будем перебирать все делители числа m: пусть g — делитель числа m, тогда $x = g - 1$.

Асимптотика такого решения $O(\sqrt{m})$, так как чтобы перебрать все делители g числа m, достаточно перебирать их до корня ($g \leqslant \sqrt{m}$).

Подзадача 5

Составим следующие уравнения:

$$\begin{cases} x + y = k \\ (x + 1) \cdot (y + 1) = m \end{cases}$$

Выразим переменную $y = k - x$ из первого уравнения и подставим во второе, получим:

$$(x + 1) \cdot (k - x + 1) = m$$

Раскроем скобки и приведем к стандартному виду:

$$x \cdot k - x^2 + x + k - x + 1 = m$$

$$-1 \cdot x^2 + k \cdot x + (k - m + 1) = 0$$

Это квадратное уравнение относительно переменной x вида:

$$A \cdot x^2 + B \cdot x + C = 0$$

где коэффициенты равны соответственно $A = -1$, $B = k$, $C = k - m + 1$.

Вычислим дискриминант квадратного уравнения по известной формуле:

$$D = B^2 - 4 \cdot A \cdot C = k^2 - 4 \cdot (-1) \cdot (k - m + 1) = k^2 + 4 \cdot (k - m + 1)$$

Если полученный дискриминант меньше нуля $D < 0$ или не существует целого числа q, такого что выполняется равенство $q^2 = D$, то значит поле нельзя разрезать требуемым образом.

В противном случае вычислим целое число $q = \sqrt{D}$, а затем найдем два решения квадратного уравнения x_1 и x_2, и соответствующие им y_1, y_2:

$$x_1 = \frac{-B - \sqrt{D}}{2 \cdot A} = \frac{k + q}{2}, \quad y_1 = k - x_1$$

$$x_2 = \frac{-B + \sqrt{D}}{2 \cdot A} = \frac{k - q}{2}, \quad y_2 = k - x_2$$

.

Остается проверить, какое из решений $\{x_i, y_i\}$, где $i = \{1, 2\}$, подходит под ограничения:

$$0 \leqslant x_i < a \quad \text{и} \quad 0 \leqslant y_i < b \quad \text{и} \quad x_i, y_i \in \mathbb{Z}$$

и выбрать его в ответ. В случае, если под ограничение подходят оба решения, необходимо выбрать в ответ то из них, где количество горизонтальных разрезов x_i минимально.

Асимптотика такого решения $O(1)$.

Задача 2. Произведение Фибоначчи

Автор задачи: Андрей Станкевич

Сделаем несколько ключевых наблюдений. Во-первых, числа Фибоначчи растут с экспоненциальной скоростью. Существует лишь 85 чисел Фибоначчи, больших 1 и не превышающих 10^{18}. Во-вторых, числа Фибоначчи имеют достаточно мало общих делителей. На самом деле можно доказать, что если нумеровать числа Фибоначчи с 1 ($F_1 = 1, F_2 = 1, F_3 = 2, \ldots$), то $\text{НОД}(F_a, F_b) = F_{\text{НОД}(a,b)}$.

Подзадача 1

Все числа Фибоначчи, не превышающие 100, которые нас интересуют: $2, 3, 5, 8, 13, 21, 34, 55, 89$. Можно либо в ручную разобрать все вариант, либо применить динамическое программирование, аналогичное решению подзадачи 2.

Подзадача 2

Будем использовать динамическое программирование. Обозначим как $d[n][k]$ количество способов представить n как произведение чисел Фибоначчи, где используются числа с номерами не больше k. Тогда:

$$d[n][k] = \sum_{i \leqslant k; n \% F_i = 0} d[n / F_i][i].$$

Используя эту формулу, мы можем найти ответ за $O(nk)$, где k — количество чисел Фибоначчи, не превышающих n, как мы уже упоминали, оно небольшое.

Подзадача 3

Заметим, что лишь три числа Фибоначчи являются степенями двойки: 1, 2 и 8. Это можно доказать, либо сгенерировать все и убедиться, что других степеней двойки среди чисел Фибоначчи до 10^{18} нет.

Значит количество способов представить число 2^k в виде произведения чисел Фибоначчи зависит от количества чисел 8 в произведении, их может быть не больше $\lfloor k/3 \rfloor$. Таким образом ответ $\lfloor k/3 \rfloor + 1$.

Подзадача 4

Заметим, что в динамическом программировании, которое решает подзадачу 2, нас интересуют только значения, которые соответствуют делителям n. Делители n можно найти за время $O(\sqrt{n})$, далее для хранения значений динамики можно воспользоваться, например, `std::map`. Таким образом, время работы получается $O(\sqrt{n}+dk)$, где d — количество делителей n (оно, очевидно, не больше $2\sqrt{n}$, а на самом деле для чисел до 10^9 не превышает 1400).

Подзадача 5

Удивительно, но на первый взгляд менее эффективный чем динамическое программирование, метод рекурсивного перебора полностью решает нашу задачу для достаточно больших n. Используя интуитивное понимание, что количество способов мало, напишем рекурсивную функцию подсчета числа произведений:

```
long long bt(long long n, int p) {
    if (n == 1) return 1;
    if (p >= fib.size()) return 0;
    if (n < fib[p]) return 0;

    long long res = bt(n, p + 1);
    if (n % fib[p] == 0) {
        res += bt(n / fib[p], p);
    }
    return res;
}
```

Задача 3. Робот-пылесос

Автор задачи: Елена Рогачева

Подзадачи 1 и 2

Для решения первых двух подзадач достаточно завести множество, хранящее все клетки, посещенные роботом. Для этого можно использовать структуру данных «`std::set`». Каждое перемещение робота на a_i можно представить как a_i перемещений на 1 в соответствующем направлении, и после каждого перемещения добавлять все k^2 покрытых роботом клеток в множество. Это решение работает за $O(nk^2 \sum a_i)$.

Подзадача 3

В третьей подзадаче решение для подзадач 1 и 2 работает за время $O(nk^2)$ и не проходит по времени. Однако можно заметить, что при каждом перемещении робота на 1 в любом направлении появляется не более k новых клеток, которые необходимо добавить в множество — полоса $1 \times k$ или $k \times 1$, в направлении которой робот сдвинулся. Таким образом, добавляя в множество только потенциально новые клетки, получаем решение третьей подзадачи, работающее за время $O(nk)$.

Подзадача 4

В этой подзадаче робот каждый раз перемещается на $a_i = k$. Разобъём плоскость на квадраты размера $k \times k$ и будем воспринимать каждый квадрат как самостоятельную клетку, «сжав» плоскость в k раз по каждой координате. На такой плоскости достаточно решить задачу о роботе размера 1×1, который каждый раз перемещается на расстояние, равное $a_i = 1$. Такая задача тоже решается «наивным» решением с множеством всех посещенных клеток. В конце достаточно умножить полученный ответ на k^2, потому что каждая «клетка» новой плоскости на самом деле состоит из k^2 клеток исходной плоскости.

Подзадача 5

В пятой подзадаче $k = 1$, но робот может перемещаться на большие расстояния. Просимулируем перемещения робота и выпишем отдельно горизонтальные и вертикальные «полосы» перемещений. Сгруппируем все вертикальные полосы с одинаковой X-координатой вместе, и сгруппируем все горизонтальные полосы с одинаковой Y-координатой вместе.

Количество посещенных клеток можно посчитать как объединение всех полос минус количество их пересечений. Объединить все полосы с общей координатой можно с помощью сортировки и одного линейного прохода (например, $[l_1, r_1]$ и $[l_2, r_2]$ с $l_1 \leqslant l_2 \leqslant r_1$ объединяются в $[l_1, r_2]$, и так далее). Найти все пересечения полос в данной подзадаче можно за время $O(n^2)$, вручную проверив на пересечение каждую пару из вертикальной и горизонтальной полос. Полосы $[l, r]$ на координате y и $[d, u]$ на координате x пересекаются, если $l \leqslant x \leqslant r$ и $d \leqslant y \leqslant u$.

Подзадача 6

В подзадаче 6 перемещения робота больше не разбиваются на достаточно небольшое количество полос ширины или высоты 1. В этой подзадаче необходимо сделать ключевое наблюдение для полного решения задачи. При каждом перемещении клетки, посещаемые роботом, образуют прямоугольник. Например, если левый-нижний угол робота находится в точке с координатами $(0, 0)$, и совершается перемещение на a_i вверх, робот посетит все клетки в прямоугольнике с углами в $(0, 0)$ и $(k, a_i + k)$. Для того, чтобы найти количество убранных клеток, достаточно найти площадь объединения таких прямоугольников.

Это можно сделать с помощью метода сканирующей прямой за время $O(n^2)$. Сожмем координаты. Заведем для каждого прямоугольника события его «начала» (левая сторона) и «конца» (правая сторона), отсортируем их по X-координате, и обработаем по очереди. В полосе между двумя событиями на X-координатах x_1 и x_2 прямоугольники покрывают одно и то же множество Y-координат клеток, так что достаточно добавить к ответу $(x_2 - x_1) \cdot w$, где w равно высоте объединения прямоугольников, покрывающих эту полосу. После каждого события w можно пересчитать за время $O(n)$.

Для решения последних двух подзадач потребуется ускорить решения подзадач 5 и 6, соответственно.

Подзадача 7

Научимся в решении для пятой подзадачи для каждой вертикальной полосы находить количество пересекающих ее горизонтальных за время $O(\log n)$. Это можно сделать с помощью дерева отрезков. Используем метод сканирующей прямой, прямая перемещается по увеличению X-координаты, а в Y-координатах, соответствующих горизонтальным полосам, храним 0, если сейчас в соответствующей координате есть полоса, либо 0, если её нет. Тогда события — это начала и концы горизонтальных полос, а также вертикальные полосы. Для события вертикальной полосы необходимо найти сумму в дереве отрезков в соответствующем ей отрезке Y-координат.

Подзадача 8

Полное решение заключается в объединении прямоугольников на плоскости. Это достаточно классическая задача, которая имеет много аналогичных решений с использованием сканирующей прямой и дерева отрезков.

Одно из решений: будем хранить в дереве отрезков для каждой Y-координаты, сколько прямоугольников ее покрывают, и возвращать количество Y-координат, покрытых хотя бы одним прямоугольником. Альтернативный подход: используем дерево отрезков на минимум и суммарный вес минимумов, тогда если минимум равен 0, то это означает, что полоса не покрыта прямоугольниками. Так можно посчитать суммарную непокрытую площадь в некотором достаточно большом объемлющем прямоугольнике, а далее вычесть ее из его площади.

Решение с деревом отрезков и сканирующей прямой работает за $O(n \log n)$.

Отметим в заключение, что координаты, в которых может оказываться робот, могут достигать $10^5 \cdot 10^9 = 10^{14}$, но площадь каждого посещенного прямоугольника не больше $10^4 \cdot 10^9 = 10^{13}$, а значит их общая площадь не превышает $10^5 \cdot 10^4 \cdot 10^9 = 10^{18}$ и помещается в 64-битном типе данных. Однако следует с осторожностью относится к промежуточным значениям, чтобы не произошло переполнения 64-битного типа данных.

Задача 4. Разноцветные точки

Автор задачи: Андрей Станкевич

Для начала рассмотрим идеи, общие для некоторых подзадач.

Во-первых, рассмотрим ориентированный граф, в котором сопоставим вершину каждой паре (i, j) $(i \neq j)$, где i — номер начальной точки, а j — номер следующей точки. Таким образом, получится $\mathcal{O}(n^2)$ вершин. Из каждой вершины графа, соответствующей паре (i, j), проведем единственное ребро в вершину, соответствующую паре (j, k), где k — номер прицельной точки для начальной точки i и следующей точки j. Таким образом, из каждой вершины графа исходит ровно одно ребро, а в терминах построенного графа процесс, описанный в условии задачи, выглядит как перемещения по ребрам.

Во-вторых, научимся определять цвета точек, пользуясь построенным графом.

Утверждение 1. Точка i будет окрашена в зеленый цвет тогда и только тогда, когда в построенном графе существует вершина, соответствующая паре (i, j) для некоторого j, которая лежит на некотором цикле.

Действительно, если такая вершина существует, то вершина i будет зеленой, потому что можно выбрать точку j в качестве следующей, и точка i побывает начальной бесконечное количество раз. В противном случае не существует такой точки j, что вершина, соответствующая паре (i, j) лежит на каком-то цикле. Так как граф конечен, а процесс продолжается до бесконечности, рано или поздно, перемещаясь по ребрам, мы попадем в какой-либо цикл и будем обходить его до бесконечности. При этом на этом цикле нет вершин, которые соответствуют состояниям, в которых i — начальная точка. Поэтому точка i не будет окрашена в зеленый цвет.

Утверждение 2. Точка i будет окрашена в синий цвет тогда и только тогда, когда она не была окрашена в зеленый цвет и существуют такие вершины v_1 и v_2, соответствующие парам (i, j_1) и (i, j_2), что вершина v_2 достижима по ребрам из вершины v_1.

Действительно, если такие вершины существуют, то вершина i будет синей, потому что можно выбрать точку j_1 в качестве следующей, и в момент, когда процесс окажется в вершине v_2 графа, точка i снова станет начальной, а точка j_2 — следующей. В противном случае, так же как и в предыдущем утверждении, процесс рано или поздно дойдет до некоторого цикла и будет обходить его до бесконечности, не оказавшись снова в состоянии, когда точка i является начальной.

Все остальные вершины будут окрашены в красный цвет.

В зависимости от оптимальности алгоритма построения графа, а также от проверки описанных критериев для определения цветов точек, решение получает различное количество баллов. Теперь рассмотрим подробно решение конкретных подзадач.

Подзадача 1

Рассмотрим построение графа. Так как все точки расположены на одной прямой, это делается достаточно просто. Для начала отсортируем все точки вдоль прямой. Для этого, например, можно упорядочить точки по возрастанию x-координат, а при равенстве — по возрастанию y-координат. Также для каждой точки сохраним ее номер в порядке сортировки.

Теперь рассмотрим начальную точку i и следующую точку j. Пусть номер точки i в порядке сортировки равен pos_i, а номер точки j в порядке сортировки равен pos_j. Рассмотрим случай, когда $pos_i < pos_j$, обратный случай рассматривается аналогично. Порядок, в котором точки упорядочены по углу относительно вектора $\overrightarrow{P_iP_j}$ выглядит следующим образом: $pos_j + 1, \ldots, n, pos_j - 1, \ldots, 1$ (имеются ввиду номера точек в порядке сортировки). Возьмем точку, находящуюся на t-й позиции в этом списке и проведем соответствующее ребро в графе. Данная часть решения работает за $\mathcal{O}(n \log n + n^2)$.

Теперь рассмотрим процесс определения цветов точек. Его можно реализовать наивно, пользуясь критериями, описанными выше.

Для того, чтобы проверить, что точка i является зеленой, переберем номер следующей точки j и проверим, что вершина, соответствующая паре (i, j), лежит на цикле. Для этого будем переходить по ребрам до тех пор, пока не попадем в очередной раз в вершину (i, j), либо не совершим n^2 переходов по ребрам. Если спустя n^2 переходов по ребрам мы ни разу не оказались в вершине (i, j), то она не лежит на цикле, так как количество вершин в графе меньше, чем n^2.

Для того, чтобы проверить, что точка i является синей, будем делать аналогичные действия. В этом случае нужно остановить процесс перехода по ребрам, если текущая вершина соответствует некоторой паре (i, k) $(k \neq j)$.

Данная часть решения работает за $\mathcal{O}(n^4)$.

Подзадача 2

Данная подзадача отличается от предыдущей ограничением на количество точек, поэтому необходимо оптимизировать вторую часть решения.

Как было описано ранее, построенный граф обладает интересным свойством — из каждой вершины исходит ровно одно ребро. Можно заметить, что такие графы выглядят как набор непересекающихся циклов, в которые «врастают» деревья.

Теперь, пользуясь этим наблюдением, выделим все зеленые точки. При помощи обхода в глубину можно найти все циклы в графе. Теперь, если некоторая вершина (i, j) лежит на цикле, покрасим точку i в зеленый цвет. Данная часть решения работает за $\mathcal{O}(n^2)$.

Осталось найти все синие точки. Для этого научимся проверять, является ли точка i синей, за $\mathcal{O}(n)$, тогда суммарно все проверки будут работать за $\mathcal{O}(n^2)$.

Выделим все вершины графа, соответствующие парам (i, j), для фиксированного i. Таких вершин $\mathcal{O}(n)$. Теперь нужно быстро проверить, существуют ли две выделенные вершины, одна из которых достижима из другой. Таким вершины не могут лежать на циклах, так как циклы графа мы обработали ранее. Поэтому такие вершины могут лежать только на деревьях. Более того, эти вершины должны лежать в одном и том же дереве, так как деревья не пересекаются друг с другом. Нетрудно понять, что в этом случае одна из вершин должна являться предком другой в дереве.

Таким образом, мы получили следующую классическую задачу: дано множество вершин, нужно определить, существует ли пара вершин, в которой одна из вершин является предком другой. Данная задача решается следующим образом. Запустим обход в глубину от корней всех деревьев и вычислим время входа и время выхода для каждой вершины. Упорядочим вершины по возрастанию времени входа. Теперь для каждой пары соседних вершин в полученном порядке нужно проверить, является ли одна из них предком другой. Нетрудно понять, что вершина v_1 является предком v_2 тогда и только тогда, когда $tin_{v_1} < tin_{v_2} < tout_{v_2} < tout_{v_1}$, где tin — время входа, а $tout$ — время выхода.

Подзадачи 3 — 6

Данная подзадача отличается от предыдущих тем, что теперь точки не лежат на одной прямой. Поэтому нужно изменить первую часть решения и оставить без изменений вторую часть решения.

Построим граф следующим образом. Зафиксируем начальную точку i и следующую точку j. Построим векторы из точки j во все остальные точки и отсортируем полученный набор векторов по углу. Также для удобства можно добавить в данный набор вектор $\overrightarrow{P_iP_j}$. Для сортировки векторов по углу можно воспользоваться функцией `atan2`, но для того, чтобы все вычисления производились в целых числах, можно воспользоваться другим способом. При сравнении двух векторов будем сначала сравнивать их координатные четверти. Если у векторов v_1 и v_2 совпадают координатные четверти, то отсортируем их по предикату $[v_1, v_2] > 0$, где $[v_1, v_2]$ — косое произведение. В случае, если косое произведение равно нулю, отсортируем векторы по квадрату их длины.

После сортировки найдем вектор $\overrightarrow{P_iP_j}$ в полученном списке. Пусть позиция этого вектора равна pos. Тогда вектор, направленный в прицельную точку, будет иметь номер $(pos + t) \pmod n$ в полученном списке. Данный способ построения графа работает за $\mathcal{O}(n^3 \log n)$.

В зависимости от того, насколько оптимально реализована вторая часть решения, данное решение может проходить различный набор подзадач.

Подзадача 7

В данной подзадаче все точки являются вершинами выпуклого многоугольника. Оказывается, что в таком случае все точки являются зелеными.

Для доказательства этого факта достаточно заметить, то для начальной точки i и следующей точки j прицельной точкой будет такая точка k, которая является t-й в порядке обхода многоугольника, начиная с точки j, вне зависимости от точки i. Значит получится следующий процесс: $(i, j) \to (j, j + t) \to (j + t, j + 2t) \to \ldots$ (номера всех точек следует брать по модулю n). Рано или поздно процесс окажется в состоянии $(j + s \cdot t, j + (s + 1) \cdot t)$, где $j + s \cdot t \equiv i \pmod n$. Осталось показать, что всегда можно выбрать точку j так, что данное уравнение имеет решение относительно s. Действительно, $s \cdot t \equiv i - j \pmod n$, поэтому можно выбрать j таким образом, чтобы было верно равенство $i - j \equiv t \pmod n$.

Подзадача 8

Для решения данной подзадачи осталось оптимизировать процесс построения графа. Научимся делать это за $\mathcal{O}(n^2 \log n)$. Для этого рассмотрим некоторую следующую точку j. Отсортируем векторы из точки j во все остальные точки по углу при помощи способа, описанного выше. Теперь переберем начальную точку i и научимся быстро находить номер прицельной точки k. Для этого воспользуемся двоичным поиском и найдем в отсортированном по углу списки векторов первый вектор, угол которого больше либо равен, чем угол вектора $\overrightarrow{P_iP_j}$. Пусть номер этого вектора равен pos. Тогда номер вектора, ведущего в прицельную точку, равен $(pos + t) \pmod n$.

Комбинируя эту часть решения с оптимальной второй частью решения, можно решить задачу на 100 баллов. Итоговое решение работает за $\mathcal{O}(n^2 \log n)$.

Задача 5. Метрострой

Автор задачи: Андрей Станкевич

Подзадача 1

Заметим, что если двигатель всего один, то существует два случая: необходимая мощность достигается при работе двигателя в первом режиме или во втором. При этом легко понять, какой режим необходим, если сравнить z_1a_1 и p. Таким образом, решение первой подзадачи выглядит так:

```
if p <= z[0] * a[0]:
    print(ceil(p / a))
```

```
else:
    print(z + ceil((p - z[0] * a[0]) / b)
```

Подзадача 2

Ограничения этой подзадачи позволяли явно перебрать и найти минимальный x, удовлетворяющий условию задачи.

```
x = 1
while True:
    q = 0
    for i in range(n):
        if x <= z[i]:
            q += a[i] * x
        else:
            q += a[i] * z[i] + b[i] * (x - z[i])
    if q >= p:
        break
    c += 1

print(x)
```

Подзадача 3

Третья подзадача также предполагала два случая: необходимую мощность можно получить, если двигатели работают в первом режиме, или необходим второй режим.

Первый случай выполняется, если $\sum\limits_{i=1}^{n} a_i z_1 \geqslant p$. В таком случае, ответ — $\dfrac{p}{\sum\limits_{i=1}^{n} a_i}$, округленное вверх.

Иначе, ответ можно найти как $\dfrac{p - \sum\limits_{i=1}^{n} a_i z_1}{\sum\limits_{i=1}^{n} b_i}$.

Подзадача 4

Будем считать, что $z_1 \leqslant z_2$ (если это не так, мысленно перенумеруем двигатели). Тогда:

1. либо оба двигателя будут работать в первом режиме ($x \in [0, z_1)$), тогда ответ — $\text{ceil}\left(\dfrac{p}{a_1 + a_2}\right)$;
2. либо двигатель 1 будет работать во втором режиме, а двигатель 2 — в первом ($x \in [z_1, z_2)$), тогда ответ — $\text{ceil}\left(\dfrac{p - (a_1 + a_2) \cdot z_1}{b_1 + a_2}\right)$;
3. либо оба будут прокладывать туннель во втором режиме ($x \in [z_2, \infty)$), ответ в этом случае — $\text{ceil}\left(\dfrac{p - (a_1 + a_2) \cdot z_1 - (b_1 + a_2) \cdot (z_2 - z_1)}{b_1 + b_2}\right)$.

Давайте для каждого случая почитаем минимальный x и выберем первый случай, в котором x попадает в заданный полуинтервал возможных значений x.

Здесь намеренно не рассматривается отдельно случай, когда $z_1 = z_2$, это возможно так как не существует таких x, которые попадают в полуинтервал $[z_1, z_2) = \varnothing$.

Полное решение, способ 1

Давайте обобщим решение предыдущей подзадачи на $n > 2$. Отсортируем двигатели по z_i. Точно так же у нас будет $n+1$ непересекающийся полуинтервал, объединение таких полуинтервалов будет давать числовой луч $[0, +\infty)$.

В каждом интервале $[z_i, z_{i+1})$ (считаем, что $z_0 = 0$ и $z_{n+1} = \infty$) можно находить минимальный x при котором достигается необходимая мощность, как $\dfrac{p - S(i)}{\sum\limits_{i=1}^{j-1} b_i + \sum\limits_{i=j}^{n} a_i}$, округленное вверх. Функцией $S(i)$ здесь обозначена мощность, которая достигается при $x = z_i$, ее можно найти по формуле $\sum\limits_{k=1}^{i-1} (z_k - z_{k-1}) \left(\sum\limits_{i=1}^{k-1} b_i + \sum\limits_{i=k}^{n} a_i \right)$.

Таким образом, в решении нужно последовательно рассматривать интервалы до тех пор, пока не найдем x, дающий достаточную мощность такой и лежащий в этом интервале.

Время работы такого решения $O(n^2)$ или $O(n \log n)$, если немного оптимизировать.

Полное решение, способ 2

Заметим, что мощность, которую дают двигатели, монотонно возрастает при увеличении x. Поэтому для нахождения минимального x можно воспользоваться бинарным поиском, на каждой итерации проверяя, в каком режиме работает каждый из двигателей.

Время работы такого решения $O(n \log n)$.

Задача 6. Красивые последовательности

Автор задачи: Рита Саблина

Подзадача 1

Для решения данной подзадачи можно было воспользоваться полным перебором. Для каждой позиции переберем, чему равен элемент (1 или 2), после чего наивно проверим выполнение условия для всех пар индексов. Асимптотика решения составит $\mathcal{O}(2^n \cdot n^2)$.

Общая идея решения

Общая идея решения следующих подзадач заключается в использовании динамического программирования. Пусть мы набрали префикс красивой последовательности, и хотим добавить в его конец элемент $x \in A$. Тогда при переходе нам необходимо и достаточно потребовать, чтобы последнее вхождение x на префиксе было хотя бы на расстоянии $x - 1$ от конца префикса.

Подзадачи 2 — 4

Определим $dp[i][j]$ как число красивых последовательностей длины i, в которых последнее вхождение числа k было на расстоянии j от конца префикса. Тогда для пересчета динамики достаточно рассмотреть два варианта: мы добавляем элемент 1 или элемент k.

Первый переход возможен всегда, и увеличивает расстояние до последнего вхождения на единицу. Второй переход возможен, если $j \geqslant k - 1$, и в результате него расстояние до последнего вхождения k становится равным нулю. Не забудем, что все арифметические операции нужно выполнять по модулю. Таким образом, псевдокод перехода в динамике выглядит так:

```
dp[i + 1][j + 1] += dp[i][j]
if (j >= k - 1) {
    dp[i + 1][0] += dp[i][j]
}
```

Таким образом, получаем динамическое программирование с n^2 состояниями, и $\mathcal{O}(1)$ переходов из каждого. Тогда асимптотика такого решения составит $\mathcal{O}(n^2)$. Заметим, что во 2 подзадаче максимальный ответ небольшой, поэтому число последовательностей можно было считать рекурсивно или перебором с отсечениями по ходу рекурсии.

Подзадачи 5 — 6

Обозначим за C максимальное ограничение на a_i ($C = 5$ в 5 подзадаче, и $C = 8$ в 6 подзадаче).

Для полного решения задачи обобщим решение предыдущих подзадач. В состоянии динамического программирования будем для каждого числа x от 1 до хранить расстояние до его последнего вхождения на префиксе, обозначим его за $d[x]$. Заметим следующий факт: если в какой-то момент расстояние для числа x стало больше, чем $x-1$, то мы можем считать, что оно равно $x-1$, так как такого расстояния до предыдущего вхождения достаточно. Таким образом, количество возможных массивов расстояний $d[x]$ не превышает $1 \cdot 2 \cdot \ldots \cdot C = C!$.

Переход в данном динамическом программировании выглядит так: переберем, какой элемент $x \in A$ мы добавим к префиксу. Тогда такой переход возможен, если $d[x] \geqslant x - 1$, и в результате перехода происходит следующее: для всех $y \neq x$ их значение $d[y]$ увеличивается на единицу, а значение $d[x]$ становится равным нулю.

Для реализации данного решения нужно хранить отображение из массива расстояний $d[x]$ в количество последовательностей в ассоциативной структуре данных. Например, на языке C++ подойдет «std::map», или же «std::unordered_map», если реализовать хеш-функцию для массива расстояний.

Получаем решение динамическим программированием с $\mathcal{O}(n \cdot C!)$ состояний, где переход выполняется за $\mathcal{O}(C^2)$. Заметим, что на самом деле достижимых состояний значительно меньше, так как, например, в достижимых состояниях все значения $d[x] < x - 1$ различны (для $C = 8$ их количество не превышает 4140). Тогда решение можно ускорить, если не делать переходы из состояний, количество последовательностей для которых равно нулю. В зависимости от эффективности решения и наличия данной оптимизации решения могли проходить только пятую подзадачу.

Задача 7. Камни

Автор задачи: Владимир Новиков

Подзадача 1

Заметим, что на каждой итерации множество камней, покрашенных в белый цвет, является последовательным отрезком. Перебрав стартовую позицию i и двигая отрезок в соответствии с условием задачи будем увеличивать счетчик ответа, если текущая длина отрезка равняется k и p-й камень был последним перекрашенным в белый цвет.

Подзадача 2

Нет смысла заново запускать алгоритм расширения отрезка после каждого запроса. Будем поддерживать массив ответа $ans[p][k]$. Достаточно заранее запустить алгоритм для каждого стартового i и на k-й итерации смотреть на последний добавленный камень p'. В таком случае увеличим $ans[p'][k]$ на единицу и продолжить выполнение алгоритма. Иначе говоря, заранее выполним передвижения отрезков для всех стартовых позиций, такой предподсчет будет работать за $O(n^2)$, а ответы на запросы за $O(1)$ — достаточно обратиться к нужной ячейке массива ответа.

Подзадача 4

В четвертой подзадаче достаточно заметить, что почти всегда мы двигаем левую границу отрезка (за исключением случая, когда левая граница отрезка достигла конца массива). Простым разбором случаев замечаем, что к ответу добавляется 1, когда $p + k - 1 < n$ и в ответ добавляется p, если

$p+1 == k$. Первый случай соответствует ситуации, когда отрезок постоянно двигает левую границу, а второй соответствует ситуации, когда мы уже не можем двигать левую границу.

Подзадачи 3, 5 и 6

Для решения следующих подзадач посмотрим на отрезок, который будет после выполнения k операций, если k-м элементом был добавлен элемент p. Тогда этот отрезок имеет вид $[p-k+1, p]$ либо $[p, p+k-1]$. Так как данные случаи симметричны, в разборе будем описывать лишь случай с $[p, p+k-1]$.

Посмотрим на позицию максимума m на данном отрезке.

Рассмотрим три случая:

1. Стартовый камень $i < m$. Иными словами, i лежит левее m. Но мы знаем, что a_m — максимальный элемент отрезка. Из этого следует, что элемент с номером m будет добавлен в расширяющийся отрезок после элемента p. Получаем, что в такой ситуации p никак не может быть добавлен в отрезок k-м.

2. Стартовый камень $i = m$. В таком случае достаточно проверить, что p является вторым максимумом отрезка и то, что $p + k$ будет добавлен позже p (иными словами $a_{p+k} > a_p$). Если данные условия выполняются, то p будет добавлен в отрезок на k-й итерации.

3. Стартовый камень $i > m$. В таком случае, если $a_{p+k} < a_m$, то мы выйдем за границы нашего отрезка $[p, p+k-1]$ до того, как добавим p в ответ. В случае же, если $a_{p+k} > a_m$, мы всегда добавим a_{p+k-1} в отрезок, а далее будем добавлять элементы со стороны левой границы, так как a_{p+k} будет больше всех элементов отрезка. Остается заметить, что все это будет происходит вне зависимости от выбора i при условии, что $i \in [m+1, p+k-1]$.

В зависимости от эффективности метода поиска максимума и второго максимума на всех отрезках длины k, мы получаем решение для различных подзадач.

Для полного решения необходимо использовать достаточно эффективную структуру данных для максимума на отрезке, например дерево отрезков, разреженные таблицы, максимум в окне любым эффективным способом.

Итоговая асимптотика: $O(n \log n + q)$ или $O(n + q)$.

Задача 8. Обыкновенная задача про строки

Автор задачи: Александр Бабин

Общие замечания.

Перед дальнейшими рассуждениями рассмотрим случай, когда длина строки s равна 1. Понятно, что при таком исходе ответ будет равен 3, так как строки «`a`», «`b`» и «`c`» попарно эквивалентны.

Также активно будет использоваться идея, о «сокращении» строки. Сокращением строки s будем называть такую строку $\eta(s)$, которая получается при сокращении блоков из одинаковых символов в строке s. Например, $\eta($«abaacccca»$) =$ «abaca». Нетрудно убедиться, что если s и t эквивалентны, то эквивалентна и пара строк $\eta(s), \eta(t)$.

Количество подстрок «`aa`» у строки s будет обозначать, как k_a, аналогично определим k_b и k_c. Такие подстроки будем в дальнейшем называть повторами символа «`a`», «`b`» или «`c`», соответственно.

Подзадача 1.

Пускай строка s имеет длину 2 и больше, а также состоит только из символов «`a`» и «`b`». Тогда $\eta(s)$ состоит из чередующихся символов «`a`» и «`b`». Рассмотрим несколько случаев:

- Если $\eta(s)$ имеет четную длину, то среди подстрок длины 2 у нее будут только «`ab`» и «`ba`». При чем одна из них будет встречаться на единицу чаще чем другая. Таким образом легко видеть, что $\eta(s)$ эквивалентна только по отношению к самой себе.

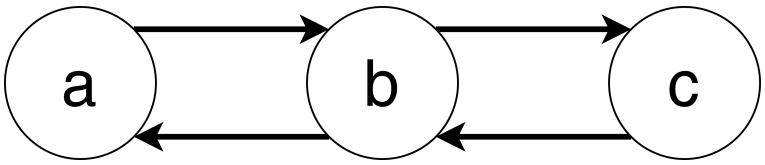
- Если $\eta(s)$ имеет нечетную длину, то среди подстрок длины 2 у нее будут только «ab» и «ba», которые встречаются одинаковое количество раз. Таким образом строке $\eta(s)$ эквивалентны две строки такой же длины, у которых также чередуются символы «a» и «b». Например, если $\eta(s) =$ «ababa», то ей эквивалентны строки «ababa» и «babab».

Нам надо посчитать количество строк t, которые эквивалентны s. Заметим, что $\eta(t)$ эквивалентна $\eta(s)$, а также, что для $\eta(s)$ можно найти не более двух эквивалентных строк. Таким образом можно перебрать $\eta(t)$. Осталось научиться обрабатывать повторы в строке t, напомним, что количество повторов в строках t и s должны совпадать, так как они эквивалентны. Но тогда задача состоит в том, чтобы для заданной строки $w = \eta(t)$ посчитать количество строк t, у которых k_a повторов символа «a» и k_b повторов символа «b». Пускай в строке w символ «a» встречается m_a раз, а символ «b» — m_b раз. Тогда в строке t будет m_a блоков из символов a с суммарной длинной $m_a + k_a$ символов и m_b блоков с суммарной длиной $m_b + k_b$. Количество строк t совпадает с количеством разбиений числа $m_a + k_a$ на m_a натуральных слагаемых и разбиений числа $m_b + k_b$ на m_b слагаемых. Таким образом, если воспользоваться методом шаров и перегородок, то понятно, что количество подходящих строк t равняется:

$$C_{m_a+k_a-1}^{k_a} \cdot C_{m_b+k_b-1}^{k_b}$$

Подзадача 2.

Решение этой подзадачи будет несколько схоже с решением предыдущей, основное отличие заключается в том, что количество строк, которые эквивалентны $\eta(s)$, может быть достаточно большим. Поэтому для начала сосредоточимся на том, какой вид они могут принимать. Если до этого мы могли избегать графической (от слова граф) сущности задачи, то сейчас сделать это уже не получится. Пускай строка «ab» встречается k_{ab} раз, строка «ac» — k_{ac} и так далее. По условию подгруппы $k_{ac} = k_{ca} = 0$. Тогда можно рассмотреть следующий ориентированный граф, состоящий из трех вершин a, b, c:



Из вершины a в вершину b будет вести k_{ab} кратных ребер, аналогичное будет верно и для любой другой пары вершин в этом графе. Тогда заметим, что последовательность вершин любого Эйлерова пути этого графа будет соответствовать какой-то строке, которая эквивалентна $\eta(s)$. При чем, если w эквивалентна $\eta(s)$, то среди символов $w_2, w_3, \dots w_{|w|}$ символ «a» встретится k_{ba} раз, символ «b» встретится $k_{ab} + k_{cb}$ раз, а символ «c» — k_{bc} раз. Таким образом, если известен первый символ строки w, то известно также и количество вхождений в него символов «a», «b», «c». Итак разберем три случая:

- $w_1 =$ «a». Этот случай имеет место быть, если Эйлеров путь может начинаться с вершины a, так происходит, если a входит в строку s хотя бы один раз, а также выполняется $s_1 =$ «a» или $s_1 = s_n$. Легко видеть, что все четные символы строки w, то есть $w_2, w_4, w_6, \dots$ равны «b». Все нечетные символы, кроме первого и последнего (который может иметь как четный, так и нечетный номер), можно выбирать свободно, при чем на этих позициях «a» встретится $k_{ab} - 1$ раз, а «b» — k_{cb} раз. Количество таких строк w равняется $C_{k_{ab}-1+k_{cb}}^{k_{cb}}$.
- $w_1 =$ «b». Аналогично первому случаю проверяется, что строка w может начинаться с символа «b». Также легко видеть, что количество подходящих строк w равняется $C_{k_{ab}+k_{cb}}^{k_{ab}}$.
- $w_1 =$ «c». Разбирается аналогично случаю $w_1 =$ «a».

Теперь можно подытожить все вышесказанное и озвучить общую схему алгоритма. Для начала мы перебираем первый символ строки w и считаем количество строк w с заданным первым символом, которые эквивалентны строке $\eta(s)$. Для каждой такой строки из заданной группы строк мы, также как и в первой подзадаче, легко можем вычислить количество строк t, эквивалентных s, таких, что $\eta(t) = w$. Так как в строках из одной группы символы «a», «b» и «c» встречаются одинаковое количество раз, то и соответствующих строк t будет одинаковое количество, таким образом можно воспользоваться правилом умножения.

С помощью предпосчитанных факториалов и обратных факториалов по модулю $10^9 + 7$ за время $O(n)$ можно разобрать каждый случай за время $O(1)$. Таким образом итоговая асимптотика алгоритма $O(n)$.

Подзадача 3.

Если $n \leqslant 13$, то достаточно научиться перебирать всевозможные строки s длиной не больше 13 и для каждой такой строки сохранить массив, состоящий из 9 чисел: количество вхождений каждой из возможной строк длины 2 в строку s. Затем можно посчитать количество вхождений каждого из этих массивов в итоговый список массивов. После вышеописанных действий можно тривиальным образом отвечать на запросы.

Тем не менее хранить и сравнивать массивы не очень то и эффективно. Поэтому все эти массивы можно занумеровать числами от 0 до $2^{21} - 1$ следующим образом. Пускай нам надо получить номер массива $a_1, \ldots, a_9$, тогда заметим, что так как $a_1 + \ldots + a_9$ не превышает 12, то ему можно сопоставить бинарную строку следующего вида:

$$1\underbrace{00\ldots00}_{a_1 \text{ нулей}}1\underbrace{00\ldots00}_{a_2 \text{ нулей}}1\ldots1\underbrace{00\ldots00}_{a_9 \text{ нулей}}$$

Длина этой строки не превышает 21, значит ей соответствует бинарное число в промежутке от 0 до $2^{21} - 1$. Легко видеть, что таким образом каждому массиву будет сопоставлено свое уникальное число.

Подзадача 4.

Вспомним подзадачу 3 и сопоставим исходной строке массив $a_1, a_2, \ldots, a_9$. Заметим, что $a_1 + \ldots + a_9 \leqslant 39$ из чего следует, что

$$U = (a_1 + 1) \ldots \cdot (a_9 + 1) \leqslant 5^6 \cdot 6^3 < 4 \cdot 10^6$$

Таким образом можно решить задачу методом динамического программирования, а именно для каждого массива $b_1, \ldots, b_9$ такого, что $0 \leqslant b_i \leqslant a_i$ и каждого символа $x \in \{a, b, c\}$ вычислить количество строк, оканчивающихся символом x, которым соответствует массив b. Такую динамику можно посчитать за время $O(U) = O((\frac{1}{9}n)^9)$.

Подзадача 5.

Немного улучшим решение предыдущей подзадачи упростив ее таким образом. Для каждого начального символа $x \in \{a, b, c\}$ вычислим количество строк, начинающихся с x и эквивалентных строке $\eta(s)$. Дело в том, что в строке $\eta(s)$ нет подстрок «aa», «bb» и «cc», а значит, ей можно сопоставить массив a длины 6, в котором встречаются количества вхождений подстрок «ab», «ac», …, «cb». Таким образом суммарное количество состояний в динамике не будет превышать:

$$U = (a_1 + 1) \ldots (a_6 + 1) \leqslant 11^4 \cdot 12^2 < 3 \cdot 10^6$$

Теперь, когда мы знаем количество строк, которые эквивалентны $\eta(s)$ и начинаются с символа x, то мы знаем, также и то, что все они будут содержать одинаковое количество букв «a» — m_a, букв «b» — m_b, и букв «c» — m_c. Но тогда мы можем также, как и в подзадаче (2) посчитать количество строк t, которые эквивалентны s и начинаются с буквы x. Итого получили решение, которое работает за время $O(U) = O((\frac{1}{6}n)^6)$.

Подзадача 6.

Попробуем воспользоваться тем, что количество строк, которые эквивалентны s не превышает 100. Тогда можно попробовать рекурсивно перебрать эквивалентные ей строки t. Сначала переберем символ t_1, в каждом из рекурсивных вызовов переберем символ t_2 и так далее. Наивный алгоритм будет работать за $O(3^n)$, но при этом, если мы сделаем отсечения, указанные ниже, то он будет работать за время $O(nw)$:

- Для каждой строки длины 2 будем поддерживать u — разность количества ее вхождений в строку s и в строку $t_1t_2 \dots t_h$ (пусть мы находимся на h-м по вложенности рекурсивном вызове). Тогда если хотя бы одно u отрицательно, то в поддереве данного рекурсивного вызова строк, которые эквивалентны строке t мы не найдем и можно сразу вернуть 0.
- Также следовало бы проверять, что оставшийся граф Эйлеров. Условие на степени вершин выполняется автоматически, его проверять не надо, но также надо проверять достижимость всех вершин, из которых выходит хотя бы одно ребро (с учетом того, что по некоторым ребрам мы уже «прошлись» в родительских рекурсивных вызовах и их учитывать не надо).

Данный алгоритм будет работать за $O(nw)$, так как в дереве перебора строк t будет ровно w листьев, которым соответсвуют строки, который эквивалентны s, и каждый такой лист будет находиться на глубине n, соответственно всего будет совершенно не более $O(nw)$ рекурсивных вызовов, каждый из которых работает за $O(1)$.

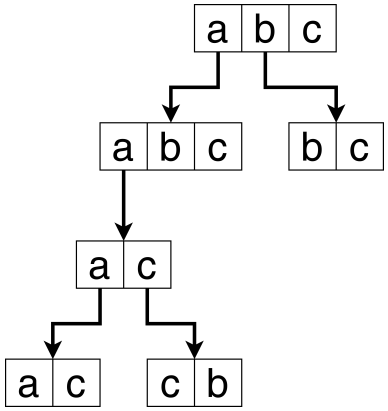
Подзадача 7.

При решении подгруппы (2) мы рассматривали строку t, которая эквивалентна строке s, как последовательность вершин в Эйлеровом обходе некоторого графа с кратными ребрами. Также как и в предыдущих группах для каждого начального символа x посчитаем количество строк w, которые эквивалентны $\eta(s)$ и уже потом с помощью полученных значения и метода шаров и перегородок решим исходную задачу. Для этого давайте попробуем «структурировать» строку w следующим образом.

Пусть w — некоторый Эйлеров обход графа, состоящего из вершин a, b, c, тогда выделим в нем самый левый подцикл и сожмем до одной вершины, будем продолжать этот процесс до тех пор, пока w не превратится в простой путь. Для наглядности приведем пример:

1. «aba<u>cbcabcabcbc</u>»;
2. «a<u>cbcabcabcbc</u>», цикл «aba» сжат в символ «a»;
3. «<u>acabcabcbc</u>», цикл «cbc» сжат в символ «c»;
4. «<u>abcabcbc</u>», цикл «aca» сжат в символ «a»;
5. «a<u>bcbc</u>», цикл «abca» сжат в символ «a»;
6. «abc», цикл «bcb» сжат в символ «b»;

В ходе этого процесса строка w превратилась в простой путь, чтобы не терять информацию об исходной строке, давайте для каждого символа, начиная с конца запомним путь, после свертки которого остался заданный символ. Всю это информацию можно представить в виде дерева, которое, например, для вышеуказанного примера выглядит так:



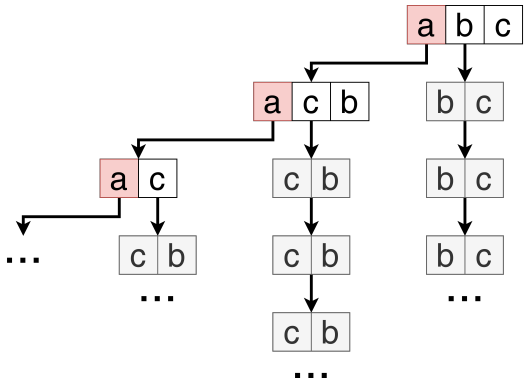
Легко видеть, что по заданному дереву легко однозначно восстановить исходную строку. Теперь давайте поговорим подробнее об этом дереве. Его корневая вершина представляет собой некоторый простой путь, а любая другая — некоторый простой цикл. В исходном графе может быть всего 5 различных циклов с точностью до циклического сдвига: «`abca`», «`acba`», «`aba`», «`aca`» и «`bcb`».

Так же легко видеть, например, что если мы рассматриваем цикл вида «xyz», который подвешен за вершину x, то в поддеревьях вершин этого цикла y и z не может присутствовать символ x. Так происходит из-за того, что при построении дерева мы всегда выделяли самый левый цикл в последовательности. Аналогично, в поддереве вершины z этого цикла не может присутствовать символ y. Но тогда поддерево вершины z пустое, так как не существует циклов, состоящих из одной вершины z.

Подобные рассуждения можно провести и при рассмотрении некоторого цикла длины 2: «xy», который подвешен за вершину x. В таком случае в поддереве y также не должны присутствовать циклы, содержащие символ x.

Теперь предположим, что мы рассматриваем строку w, начинающуюся с символа «`a`», тогда все циклы, содержащие символ «`a`» будут образовывать непрерывную цепочку, то есть вертикальный путь, который подвешен к первой вершине простого пути. Единственный оставшийся цикл «`bcb`» (или, что тоже самое «`cbc`»), может образовывать сразу несколько цепочек, которые могут быть подвешены либо за исходный простой путь, либо какую-то вершину-цикл, содержащую символ «`a`».

Приведем пример-иллюстрацию, как может выглядить дерево, построенное описанным выше образом:



Итак, каждой строке w, которая эквивалентна $\eta(s)$ и начинается с символа «`a`» мы сопоставили некоторое уникальное дерево. Это значит, что если мы вычислим количество способов составить такие деревья, то мы также найдем количество искомых строк. Как мы уже поняли дерево можно «собрать» из какого-то простого пути, который начинается с символа «`a`», а также какого-то набора циклов. При чем каждое ребро в графе должно встретиться ровно один раз либо в просто пути, либо в каком-нибудь из циклов. Обработаем каждое такое разбиение графа на путь и простые циклы отдельно:

Для начала переберем простой путь, затем вычтем его ребра из графа. Если у какой-то вершины в графе при этом количество исходящих ребер не оказалось равно количеству входящих, то с таким

простым путем собрать дерево не получится. В ином случае давайте заметим, что имеет место быть равенство:

$$k_{ab} - k_{ba} = k_{bc} - k_{cb} = k_{ca} - k_{ac} = \Delta$$

Пускай мы разбиваем граф на t_{ab} циклов «aba», t_{ac} циклов «aca», t_{bc} циклов «bcb», t_{abc} циклов «abca» и t_{acb} циклов «acba». Тогда легко видеть, что $\Delta = t_{abc} - t_{acb}$, так как $k_{ab} = t_{abc} + t_{ab}$ и $k_{ba} = t_{acb} + t_{ab}$. Переберем количество циклов t_{abc}, тогда легко видеть, что однозначно восстанавливаются и количества всех остальных циклов: $t_{acb} = t_{abc} - \Delta$, $t_{ab} = k_{ab} - t_{abc}$, $t_{bc} = k_{bc} - t_{abc}$, $t_{ac} = k_{ca} - t_{abc}$. При $\max\{\Delta, 0\} \leqslant t_{abc} \leqslant \min\{k_{ab}, k_{bc}, k_{ca}\}$ мы получаем корректное разбиение на циклы. Легко видеть, что существует всего $O(n)$ различных разбиений оставшегося графа на циклы.

Теперь у нас есть циклы, из которых мы можем собрать дерево. Введем переменную G, которая равна 1, если длина фиксированный на первом шаге простой путь состоит из двух или трех вершин, и равна 0, в ином случае. А также введем $K = G + t_{abc} + t_{acb} + t_{ab} + t_{ac}$. Для начала расставим циклы, содержащие символ «a», количество способов сделать это равняется:

$$\frac{(t_{abc} + t_{acb} + t_{ab} + t_{ac})!}{t_{abc}! \cdot t_{acb}! \cdot t_{ab}! \cdot t_{ba}!}$$

Теперь к нашему дереву стоит подвесить циклы «bcb», коих у нас t_{bc} штук. Все подвешенные циклы будут образовывать цепочки, которые могут начинаться в K различных позициях, вне зависимости от того, в каком порядке мы расставили циклы, содержащие символ «a». Так как каждый такой способ эквивалентен некоторому разбиению числа t_{bc} на K неотрицательных целых слагаемых, то это можно сделать $C^{K-1}_{t_{bc}+K-1}$ различными способами. Таким образом для каждого выбранного простого пути и каждого разбиения на циклы мы можем вычислить количество заданных деревьев за время $O(1)$, а с учетом того, что мы рассмотрим $O(1)$ простых путей, а для каждого из которых рассмотрим всего лишь $O(n)$ разбиений на циклы, то искомое количество строк w, которые эквивалентны строке $\eta(s)$ и начинаются с символа «a» мы найдем за время $O(n)$ по следующей формуле:

$$\sum_{t_{abc}, t_{acb}, \ldots} \frac{(t_{abc} + t_{acb} + t_{ab} + t_{ac})!}{t_{abc}! \cdot t_{acb}! \cdot t_{ab}! \cdot t_{ba}!} \cdot C^{K-1}_{t_{bc}+K-1}$$

Теперь, точно также как и в подгруппе (2) мы можем посчитать количество строк t, которые начинаются с символа «a» и эквивалентны строке s. Аналогично поступим и для вычисления количества строк t, которые начинаются с символа «b» или «c».