Всероссийская олимпиада школьников по информатике 2023–2024

Региональный этап

Разбор задач

Условия задач, тесты, решения и разбор задач подготовили Никита Голиков, Александр Горбунов, Мария Жогова, Евгений Пахомов, Михаил Первеев, Роман Первутинский, Маргарита Саблина, Владимир Смаглий, Андрей Станкевич, Федор Царев, Александр Чистяков, Екатерина Шиляева, Григорий Шовкопляс.

Ценные замечания по результатам тестирования задач сделали Николай Будин, Леонид Данилевич, Игорь Маркелов, Денис Мустафин, Максим Туревский.

Задача 1. Посадка в самолет

Пронумеруем места в каждом ряду слева направо с 0 до 5 включительно. Места с 0 до 2 включительно находятся слева от прохода, а места от 3 до 5 — справа. Заметим, что в ряду i у места с номером j симметричное для него относительно прохода место будет иметь координаты $i, 5 - j$.

Подзадача 1

В первой подзадаче все пассажиры прошли регистрацию онлайн, поэтому неоходимо только проверить, чтобы рассадка была симметричной относительно прохода, для этого для всех мест (i, j) значения на местах (i, j) и $(i, 5 - j)$ должны совпадать.

Подзадача 2

Изначально в самолете свободны все места, поэтому нужно симметрично рассадить пришедших на стойку регистрации. Заметим, что это не получится сделать, если m — нечетное и если количество мест в самолете меньше, чем пришедших на регистрацию пассажиров. Количество мест в самолете — $6 \cdot n$. Если условие выполняется, то рассадим каждую пару пассажиров на симметричные места (i, j) и $(i, 5 - j)$. Начнем с $i = 0$ и будем заполнять места с номерами 0 и 5, 1 и 4, 2 и 3, пока пассажиры не кончатся.

Подзадача 3

При $m = 1$ необходимо, чтобы только для одного места (i, j), занятого при онлайн-регистрации, не было пары на симметричном относительно прохода месте $(i, 5 - i)$. В этом случае такое незанятое место необходимо занять. Если незанятых симметрично при онлайн-регистрации мест больше одного, то рассадить пассажиров не получится.

Подзадача 4

Если при онлайн-регистрации было занято только одно место, то необходимо, чтобы количество пассажиров на стойке регистрации было нечетным, а также суммарное количество пассажиров $m+1$ не превысило количество мест в самолете $6 \cdot n$. Если эти условия не выполнены, то рассадку сделать нельзя. Если эти условия выполнены, то неоходимо занять симметричное для единственного, зарегистрированного онлайн, пассажира, а затем симметрично заполнить все остальные места.

Подзадача 5

Начнем заполнять места (i, j) пассажирами, начиная с $i = 0$. Сначала рассадим пассажиров на пустые места, симметричные занятым при онлайн-регистрации. Если место (i, j) занято при онлайн-регистрации, проверим место $(i, 5 - j)$. Если оно тоже занято при онлайн регистрации, то пропустим его, если же оно свободно, то займем его пассажиром со стойки. Если количество пассажиров со

стойки будет недостаточно, чтобы заполнить все пустые симметричные места, то рассадить пассажиров не получится. Если у всех пассажиров с онлайн-регистрации есть пара на симметричном месте и остались пассажиры со стойки, которых еще не посадили, рассадим их симметрично на пустые места. Если таких пассажиров осталось нечетное количество или количество онлайн-зарегистрированных пассажиров и пассажиров со стойки суммарно превышает $6 \cdot n$ — количество мест в самолете, — то рассадить пассажиров не получится. Иначе попарно рассадим оставшихся пассажиров со стойки симметрично, аналогично подзадаче 2 и 4.

Задача 2. Битоническая последовательность

Подзадача 1

Переберем все подходящие под условие (l, r) и для каждой такой пары сделаем проверку на битоничность.

Подзадача 2

Заметим, что для любых $l < r$ таких, что подотрезок с l по r является битоническим верно и то, что подотрезок с l по $r - 1$ — тоже битонический. Значит, мы можем зафиксировать левую границу и увеличивать правую, пока подотрезок удовлетворяет условию битоничности.

Подзадача 3

Заметим, что для битонической последовательности длины n ответ будет n^2. Давайте разобьем весь массив на битонические последовательности максимальной длины: как только наш отрезок перестаёт быть битоническим, начинаем новый в этом же месте Например, последовательности $[1, 2, 5, 3, 4]$ нужно разбить на отрезки

- $(1, 4)$ последовательность $[1, 2, 5, 3]$
- $(4, 5)$ последовательность $[3, 4]$

При выводе ответа, нужно не забыть вычесть количество элементов, которые попали в два подотрезка одновременно (в этой подзадаче их количество = количество разбиений минус один).

Подзадача 4

Для полного балла нужно дополнительно обработать случай, когда подряд идет несколько одинаковых чисел. Заметим, что такие числа всегда будут находится в разных подотрезках. То есть теперь количество элементов, которые попали в два подотрезка одновременно = количество разбиений − (количество рядомстоящих одинаковых чисел +1)

Задача 3. Игра с таблицей

Подзадача 1

Для решения этой подзадачи достаточно для каждого подмножества столбцов посчитать сумму чисел в них. Это решение работает за $O(2^m)$.

Подзадача 2

В этой подзадаче сумма чисел в i-й строке не превосходит i. Несложно показать по индукции, что в этом случае любую допустимую сумму можно набрать, не применяя операции над столбцами. Для этого применим следующий алгоритм: обозначим сумму чисел в i-й строке за sum_i. Будем перебирать строки в порядке уменьшения суммы чисел в них. Если сумма чисел sum_i в очередной строке не превосходит s, то вычтем из s sum_i. Иначе добавим строку i в ответ (то есть она будет удалена из таблицы).

Итоговая асимптотика такого решения составит $O(nm + n \log n)$.

Подзадача 3

Для решения этой подзадачи необходимо вручную рассмотреть все варианты того, какие строки будут удалены из таблицы. Сумма чисел в каждом из столбцов для каждого такого варианта фиксированна, поэтому можно применить решение, аналогичное решению первой подзадачи в каждом из них.

Подзадача 4

В этой подзадаче ограничения на n и m небольшие, поэтому можно было перебрать, какие строки и столицы будут удалены из таблицы, после чего посчитать сумму оставшихся элементов и сравнить её с s.

Итоговая асимптотика решения $O(2^{n+m}nm)$.

Подзадача 5

В этой подзадаче ограничения на n и m слишком большие для решения из 4й подзадачи, поэтому необходимо его улучшить. Для этого можно было применить технику «meet in the middle». Разделим таблицу на две части примерно одинакового размера, а именно, положив $k = \lfloor \frac{m}{2} \rfloor$ отнесем столбцы $1, 2, \ldots, k$ к таблице A_1 и столбцы $k+1, k+2, \ldots, m$ к таблице A_2.

Зафиксируем множество строк, которые будут удалены. Теперь сумма чисел в каждом столбце фиксированна. Заметим, что для любого множества, которое мы удалим в левой половине, мы можем выбрать произвольное множество столбцов, которые будут удалены в правой половине.

Посчитаем величины $sum\_left[mask][mask'], sum\_right[mask][mask']$: $sum\_left[mask][mask']$ равно сумме чисел, которые останутся в левой половине таблицы (таблице A_1), если множеству удаленных строк соответствует маска $mask$, а множеству удаленных столбцов в первой половине соответствует маска $mask'$. Аналогично определяется sum_right.

Зафиксируем пару $(mask, mask'_l)$ в левой половине. Тогда такой паре множеств удаленных строк и столбцов соответствует пара $(mask, mask'_r)$ в правой половине, где $mask'_r$ произвольна. Значит, так как мы хотим получить общую сумму оставшихся чисел в таблице равной S, сумма чисел в правой части таблицы должна быть равна $S - sum_left[mask][mask'_l]$. Достаточно сохранить все значения $sum_right[mask][mask'_r]$, которые можно получить для такой маски $mask$ в некоторой структуре данных, которая позволить быстро проверять наличие числа в ней. Это может быть std unordered_map, но на практике отсортированный массив в сочетании с std lower_bound оказывается сильно более эффективным, так как для фиксированной маски $mask$ необходимо хранить маленькое количество различных значений.

Итоговая асимптотика решения составляет $O(2^{n+\frac{m}{2}}nm)$.

Подзадача 6

В этой подзадаче ограничения на n и m полные, но есть дополнительное ограничение $a_{i,j} \leqslant 6$.

Зафиксируем множество столбцов, которые будут удалены. Тогда сумму в каждой строке станет фиксированной, поэтому достаточно научиться проверять, что можно выбрать подмножество строк с необходимой суммой чисел в них. Это является классической задачей о рюкзаке, поэтому несложно реализовать эту проверку за $O(n \cdot nmW)$, где W — максимальное число, которое может встретиться в таблице.

Итоговая асимптотика решения $O(2^{n+m}nmW)$.

Подзадача 7

Полное решение этой задачи аналогично решению 5й подзадачи, но требует дальнейших оптимизаций. Научимся считать $sum\_left[mask][mask']$ более эффективно. Для этого нужно быстро научиться считать сумму по подмаскам, а также перебирать маски в оптимальном порядке.

Зафиксируем маску $mask$, для каждого столбца посчитаем сумму чисел, которые в нем останутся, если будут удалены строки, соответствующие $mask$, и сохраним сумму чисел, которые останутся в i-м столбце в $sum\_left[mask][2^i]$. Несложно реализовать эту часть за $O(2^n nm)$.

Далее посчитаем сумму по подмаскам в массиве $sum\_left[mask]$. Это стандартная задача, которую можно решить за $O(2^k k)$, это будет сделано для каждой маски $mask$, поэтому это будет работать за $O(2^{n+k}k)$.

Оставшаяся часть решения аналогична подзадаче 5: мы фиксируем пару $(mask, mask'_l)$ для левой половины таблицы и при помощи std lower_bound проверяем, что для правой половины таблицы существует подходящая маска $mask'_r$.

Итоговая асимптотика такого решения составит $O(2^{n+k}k)$, где $k = \lceil \frac{m}{2} \rceil$.

Задача 4. Выбор столицы

Во всех подзадачах применима идея двоичного поиска. Пусть мы хотим найти минимальную *доступность*, добавив не более k ребер. Тогда можно сделать двоичный поиск по ответу, найти минимальное количество добавленных ребер, и сравнить его с k. Так же заметим, что оптимально проводить ребра только из столицы.

Подзадача 1

В этой подзадаче граф являлся бамбуком (иными словами, путь из 1 в n), и требовалось найти ответ только для первой вершины. В двоичном поиске нам нужно найти минимальное количество ребер, чтобы расстояние до каждой вершины было не больше x. Тогда заметим, что мы должны провести хотя бы одно ребро в вершины, начиная с вершины с номером $n - x$ (в 0-индексации), поэтому оптимально провести ребро в вершину $n - x$. Далее мы проведем ребро в вершину $n - 2x$, и так далее. Таким образом, нужно найти максимальное t, такое что $n - tx > 0$, решив линейное неравенство.

Подзадача 4

В этой подзадаче граф являлся бамбуком, но уже нужно было найти ответы для всех вершин. Для этого можно за $O(k)$ перебрать, сколько ребер мы проведем вправо, а сколько влево, и далее применить решение из первой подзадачи. Получится решение за $O(nk \log n)$.

Подзадача 2

В этой подзадаче можно было перебрать, в какую вершину мы добавим ребро, и наивно посчитать расстояния до всех вершин (например, обходом в ширину). Получится решение за $O(n^2)$.

Подзадача 5

В этой подзадаче можно было перебрать подмножество вершин, в которое мы проведем дополнительные ребра, и наивно посчитать расстояния. Получится решение за $O(2^n \cdot n^2)$.

Жадная идея решения

Для остальных подзадач рассмотрим следующую идею. Пусть мы хотим найти минимальное количество ребер, чтобы ответ не превосходил x. Рассмотрим самую далекую вершину v от s. Если расстояние до нее не превосходит x, то до всех остальных вершин тоже, и поэтому ребра проводить не нужно. Иначе, рассмотрим $(x - 1)$-го предка вершины v в дереве (иными словами, поднимемся от вершины v на $x - 1$ ребер вверх). Назовем эту вершину u. Тогда мы должны провести хотя бы одно ребро в поддерево вершины u.

Тогда докажем, что существует оптимальный ответ, в котором мы провели ребро именно в вершину u. Рассмотрим вершину w из поддерева u, в которую мы провели ребро. Тогда давайте уберем ребро $s \to w$, и добавим ребро $s \to u$. От этого количество добавленных ребер не увеличится, и все

расстояния все еще останутся $\leqslant x$, так как глубина поддерева вершины u не превосходит $x-1$, а для вершин вне поддерева мы ничего не сломали.

Тогда работает следующий жадный алгоритм: пока можно, находим самую далекую вершину v, проводим ребро в ее $(x-1)$-го предка u, и продолжаем алгоритм. Заметим, что после добавления ребра $v \to u$ про вершины в поддереве u можно забыть, поэтому на каждой итерации граф остается деревом. Так же заметим, что если мы уже провели $k+1$ ребро, то можно завершить алгоритм, поэтому для фиксированной вершины s наивно реализованный жадный алгоритм работает за $O(nk)$.

Подзадача 3

Воспользовавшись двоичным поиском и описанным выше жадным решением, получаем решение за $O(n \log n)$ для третьей подзадачи.

Подзадачи 6-7

В этой подзадаче для каждой столицы можно было перебрать ответ за $O(n)$ или за $O(\log n)$ двоичным поиском, применить описанное выше решение за $O(n^2)$, суммарно получив решение за $O(n^4)$ или $O(n^3 \log n)$.

Подзадача 8

Научимся решать задачу для фиксированного корня и верхнего ограничения на ответ за $O(n)$. Для этого заметим, что описанный выше жадный алгоритм можно реализовать, используя обход в глубину. На каждой итерации мы выбираем самую глубокую вершину, это эквивалентно тому, что мы будем при обходе в глубину сначала добавлять ребра в поддеревья детей, а затем в текущую вершину, если нужно.

Для того чтобы реализовать такой обход в глубину, нам нужно научиться понимать, когда добавлять ребра. В обходе в глубину из вершины будем возвращать целое число от 0 до $x-1$, равное максимальному расстоянию от вершины вниз. Если оказалось, что у ребенка текущей вершины расстояние равно $x-1$, то в него нужно провести ребро (кроме случая, когда текущая вершина является столицей). Итоговую глубину вершины считаем как максимум из глубин детей плюс один. Получается решение для одной столицы за $O(n)$, что суммарно по всем столицам дает $O(n^2)$.

Подзадачи 9-10

Рассмотрим полное решение задачи. Подвесим дерево за вершину с номером 1, и обсудим решение для первой вершины в качестве столицы.

Нам нужно уметь делать следующее: быстро находить самую глубокую вершину, подниматься от нее на $x-1$ вверх, и помечать все вершины на поддереве как удаленные.

Рассмотрим Эйлеров обход дерева (сначала выписывается вершина, потом рекурсивно выписываются ее поддеревья). При таком обходе поддерево любой вершины образует подотрезок обхода. Давайте поддерживать текущие расстояния до вершин в дереве отрезков по Эйлеровому обходу на максимум. Тогда, чтобы найти самую глубокую вершину, нужно взять максимум на всем массиве обхода. Чтобы пометить поддерево как удаленное, вычтем большое число из всех значений на отрезке (например, достаточно вычесть n).

Для того чтобы подняться на $x-1$ вверх, предподсчитаем двоичные подъемы на дереве, тогда за $O(\log n)$ мы можем подняться на любое расстояние вверх.

Таким образом, если мы проделаем описанный выше процесс k раз, мы получим проверку за $O(k \log n)$ для одной столицы после препроцессинга за $O(n \log n)$.

Теперь обсудим, как обобщить это решение для всех столиц. Будем считать ответы для столиц в порядке обхода в глубину по дереву. Когда мы спускаемся от вершины к ее ребенку, нужно пересчитать массив расстояний до всех вершин. Он изменяется следующим образом: для вершин на поддереве ребенка вычитается 1, а для всех остальных вершин прибавляется 1. Поэтому мы можем поддерживать текущее дерево отрезков с расстояниями до вершин, и пересчитывать его при переходе в ребенка с помощью прибавлений на отрезках.

Осталось обсудить несколько деталей. В каждой вершине мы делаем двоичный поиск по ответу, в ходе которого мы изменяем текущий массив расстояний прибавлениями на отрезке. После того как мы нашли минимальное количество ребер (или поняли, что оно превосходит k), давайте откатим все совершенные прибавления с помощью вычитаний на отрезке.

Так же заметим, что если для изначальной столицы нам нужно было подниматься на $x-1$ вверх, то для следующих столиц s нам нужно находить $(x-1)$-ю вершину на пути от текущей (самой глубокой) вершины v до текущей столицы. Для этого, давайте найдем $\mathrm{LCA}(v,s)$ (самого глубокого общего предка), и посчитаем длины путей от v до LCA и от s до LCA. Тогда мы сводим запрос к подъему вверх на одном из двух этих вертикальных путей. Поиск LCA можно реализовать любым стандартным алгоритмом поверх двоичных подъемов.

Теперь заметим, что если для изначальной столицы мы вычитали n на отрезке-поддереве, то теперь нам нужно вычитать на поддереве u, если бы текущая столица s была корнем. Есть два случая. Если вершина u не является предком s, то поддерево при подвешивании за s остается таким же, как при изначальной столице. Если u это предок s, то давайте рассмотрим вершину w на пути из s в u, родителем которой является u. Чтобы найти эту вершину, нужно подняться из s вверх на расстояние на один меньшее, чем длина пути между s и u. Теперь поддерево u при подвешивании за s это все вершины, кроме изначального поддерева вершины w, поэтому, чтобы прибавить к нему, можно прибавить на суффиксе и на префиксе обхода.

В итоге получаем решение за $O(nk\log^2 n)$: бинпоиск в каждой вершине, и проверка за $O(k\log n)$.

Для полного решения задачи нужно избавиться от бинпоиска в каждой вершине. Для этого заметим следующий факт: если вершины v и u соединены ребром дерева, то ответы для этих вершин в качестве столиц отличаются не более, чем на 1. Это верно, потому что при изменении столицы с v на u всего одно ребро меняет направление (ребро между v и u), и если мы раньше могли дойти от вершины v до вершины w за x ребер, то теперь мы можем дойти за $x+1$ ребро, пройдя изначально по ребру $u \to v$.

Тогда полное решение выглядит так: в изначальной вершине сделаем бинпоиск между 1 и $n-1$, а далее при переходе в обходе в глубину из вершины v в вершину u мы будем делать бинпоиск между $\mathrm{ans}[v]-1$ и $\mathrm{ans}[v]+1$, и он отработает за $O(1)$ проверок.

Получаем итоговое решение задачи за $O(nk\log n)$.

Задача 5. Разбиение массива

Заведем массив col — цвета элементов массива a.

Подзадача 1

Заметим, что равные числа можно покрасить в один цвет, поскольку их отношение 1 не является простым числом.

В первой подзадаче элементами массива могут быть только 1 или 2. Покрасим все $a_i = 1$ в один цвет, а $a_i = 2$ в другой, для этого можно использовать значение элемента в качестве его цвета $col_i = a_i$. Такая раскраска будет правильной, поскольку 1 и 2 будут разного цвета — их отношение простое, а равные числа можно покрасить одним цветом.

Подзадача 2

Отсортируем массив по возрастанию. Найдем число p — им будет наименьший простой делитель любого элемента массива. Покрасим a_0 в первый цвет $col_0 = 1$, далее пройдем по всем элементам массива, начиная с $i = 2$, если $a_i = a_{i-1} * p$, то покрасим a_i в цвет, отличный от a_i: $col_i = 3 - col_{i-1}$, иначе покрасим элемент a_i в такой же цвет, как и a_{i-1}: $col_i = coli - 1$. Такая раскраска будет правильной, поскольку только соседние степени числа p будут давать в отношении простое число p. Если соседние числа равны или отличаются более, чем в p раз, то их отношение — единица или p^k, где $k \geqslant 2$ — числа, не являющиеся простыми.

Подзадача 3

Элементами массива могут быть только 1, 2 или 3. Покрасим все $a_i = 1$ в один цвет $col_i = 1$, а $a_i = 2$ и $a_i = 3$ в другой $col_i = 2$. Такая раскраска будет правильной, поскольку не кратные друг другу элементы со значением 2 и 3 будут одного цвета цвета, а 1 будет противоположного цвета.

Подзадача 4

Элементами массива могут быть только 1, 2, 3 или 4. Раскрасим массив как в подзадаче 3: $a_i = 1$ в один цвет $col_i = 1$, а $a_i = 2$ и $a_i = 3$ в другой $col_i = 2$. Элементы со значением 4 покрасим в цвет 1. Такая раскраска будет правильной, поскольку не кратные друг другу элементы со значением 2 и 3 будут одного цвета цвета. 4 и 1 одного цвета, 4 нацело делится на 1, но в их отношении получается не простое число.

Подзадача 5

Для маленьких значений n задачу можно решить полным перебором: для всех раскрасок массива a в два цвета, выполняется ли условия для обоих элементов одного цвета. Для этого проверим, что для каждой пары элментов $a_i \leqslant a_j$ отношение $\frac{a_j}{a_i}$ не является простым числом, проверка выполняется до $\sqrt{\frac{a_j}{a_i}}$. Асимптотика $O(2^n n^2 \sqrt{\max(a_1, a_2, \ldots, a_n)})$

Подзадача 6

Посчитаем для каждого элемента массива a_i количество его простых делителей. Для этого заведем счетчик простых делителей, переберем все значения k от 2 до $\sqrt{a_i}$ включительно. Если $a_i \vdots k$, то будем делить a_i на k и увеличивать счётчик, пока a_i нацело делится на k. Далее заметим, что мы можем покрасить числа с четным числом простым делителей в цвет 1, а с нечетным — в цвет 2. Таким образом, если числа отличаются друг от друга в p раз, где p — простое, то количество простых делителей у таких чисел будет отличаться друг от друга на 1, таким образом, у них будет разная четность, и такая раскраска будет правильной. Итоговая асимптотика будет $O(n\sqrt{\max(a_1, a_2, \ldots, a_n)})$.

Задача 6. Бактерии

В решениях всех подгрупп опускается фраза «если ответ не был найден, выводим -1».

Подзадача 1

Исходя из ограничений, ни одна бактерия не успеет начать размножаться. Кроме того, ответ не превосходит 10^5.

Построим массив подсчёта c_j — количество бактерий, которые попадут в чашку Петри в момент времени j. Переберём ответ t. Если сумма c_j по всем $j \leqslant t$ равна m, то мы нашли ответ. Если эта сумма превысила t, выводим -1.

Асимптотика $O(n + \max(a_1, a_2, \ldots, a_n))$.

Подзадача 2

Заведём счётчик замороженных бактерий c и счётчик активных a. Переберём i от $a_1 + t_1$ до $a_1 + t_1 + 40$ ($m \leqslant 10^9 < 2^{40}$).

- Если $i \leqslant n$, то в чашку Петри попадает новая замороженная бактерия, увеличим c на 1;
- Если $c > 0$, значит одна новая бактерия разморозилась, уменьшим c на 1 и увеличим a на 1;
- Все активные бактерии размножились, умножим a на 2.

Если $c + a = m$, выводим ответ. Если $c + a > m$, выводим -1.
Асимптотика $O(n)$.

Подзадача 3

На маленьких ограничениях можно написать не очень аккуратную эмуляцию. Проверим все t от 1 до T. Научимся валидировать ответ за $O(n)$. Переберём i от 1 до n и посчитаем количество бактерий, порождаемое каждой изначальной бактерией независимо.

- Если $t < a_i$, значит бактерия не успеет попасть в чашку Петри;
- Если $a_i \leqslant t < a_i + t_i$, бактерия останется замороженной и даст вклад 1;
- Если $a_i + t_i - t > 40$, значит бактерия породит больше 2^{40} бактерий, но $m \leqslant 10^9 < 2^{40}$;
- В противном случае бактерия даст вклад $2^{a_i+t_i-t+1}$.

Посчитаем сумму вкладов бактерий. Если она равна m, мы нашли ответ, иначе выбранное t нам не подходит.

Теперь несложно получить оценку на T: $T \leqslant \max(a_1, a_2, \ldots, a_n) + \max(t_1, t_2, \ldots, t_n) + 40 \leqslant 6040$.

Асимптотика решения $O(n \cdot T)$.

Подзадача 4

В этой подзадаче все n бактерий уже находятся в чашке Петри. По утверждению из предыдущей подзадачи нам достаточно рассмотреть все t из отрезка $[\min(t_1, t_2, \ldots, t_n), \ldots, \min(t_1, t_2, \ldots, t_n)+40]$. Для каждого выполним проверку по методу третьей подзадачи за $O(n)$.

Асимптотика решения $O(n \log m)$.

Полное решение

Для решения задачи на полный балл можно аккуратно улучшить решение четвёртой подзадачи, но есть более красивый вариант.

Заметим, что функция «количество бактерий от времени» неубывающая. Сделаем бинпоиск по ответу, установим левую границу в 0, правую в $R = 2 \cdot 10^9 + 40$. Левая граница будет отвечать за время, с которым мы набираем меньше m, правая за время, с которым мы набираем хотя бы m.

Для движения границ будем использовать суммарный вклад из третьей подгруппы. Если он меньше m, будем двигать левую границу, иначе правую.

Асимптотика решения $O(n \log R)$.

Задача 7. Разбиение на тройки

Во всех подзадачах будет удобно предпосчитать массив cnt где cnt_i $(1 \leqslant i \leqslant m)$ — количество вхождений числа i в массив a.

Подзадача 1

В данной подзадаче возможен только один тип троек подряд идущих чисел — из чисел 1, 2 и 3. Заметим, что при фиксированном количестве таких троек, все остальные тройки должны состоять из одинаковых чисел. Количество троек подряд идущих можно перебрать, а далее оставшиеся количества каждого из чисел должны делиться на три, чтобы разбиваться на тройки из одинаковых чисел. Таким образом, перебором количества троек подряд идущих и проверкой того, что оставшиеся числа разбиваются на тройки одинаковых чисел, можно посчитать количество вариантов. Так как количество троек точно не больше суммарного количества чисел n, данное решение работает за $O(n)$.

Подзадача 2

В данной подзадаче возможны два типа троек подряд идущих чисел — из чисел 1, 2, 3 и из чисел 2, 3, 4. Если мы будем перебирать возможные варианты комбинаций количеств каждой из таких троек аналогично решению в подзадаче 1 и проверять, бьются ли оставшиеся числа на тройки одинаковых, получим решение за $O(n^2)$.

Подзадача 3

Заметим, что в данной подзадаче не бывает троек из одинаковых чисел. Начнем разбивать числа на тройки подряд идущих, начиная с числа 1. Заметим, что все числа 1 должны войти в тройки вида (1, 2, 3), вычтем необходимое количество таких троек из cnt_1, cnt_2 и cnt_3, и перейдем к числу 2. Аналогично заметим, что все оставшиеся числа 2 должны войти в тройки вида (2, 3, 4), так как чисел 1 уже не осталось. Продолжая выполнять такие рассуждения и действия для всех чисел от 2 до $m-2$ мы сможем единственным образом распределить все тройки. Если в процессе мы хотели уменьшить cnt_i, когда оно было равно нулю, или после всех действий $cnt_{m-1} \neq 0$ или $cnt_m \neq 0$, мы не смогли разбить числа на тройки. Если мы смогли разбить числа на тройки, так как наше разбиение строится однозначно, количество разбиений на тройки равно 1.

Подзадача 4

В этой подзадаче возможны тройки подряд идущих только следующего вида: $(x, x+1, x+2)$, где остаток от деления x на 4 равен 1. Также невозможны тройки из одинаковых чисел вида (x, x, x), где x делится на 4. Заметим, что если мы знаем количество способов независимо разбить наборы чисел равных x, $x+1$, $x+2$ для всех x с остатком от деления на 4 равным 1, то ответ для всех чисел будет равен произведению этих количеств способов. Теперь заметим, что мы уже научились считать ответ для таких наборов в подзадаче 1 за линейную асимптотику от $min(cnt_x, cnt_{x+1}, cnt_{x+2})$ для каждого набора. Нетрудно увидеть, что поиск ответа для всех наборов займет не более, чем $O(n)$, так как $cnt_1 + cnt_2 + \ldots + cnt_m = n$.

Подзадача 5

Заметим, что задачу можно решать динамическим программированием. Заведем динамику $dp_{i,\ left\_prev,\ left\_curr}$, в которой будет храниться количество способов разбить числа от 1 до i на тройки, если чисел от 1 до $i-2$ не осталось, чисел $i-1$ осталось $left\_prev$, а чисел i осталось $left\_curr$, причем все тройки $(i-1, i-1, i-1)$ уже использованы. С таким состоянием базой будет $dp_{1,\ 0,\ cnt_1} = 1$. Переходы динамики будут выглядеть следующим образом:

- Для троек одинаковых чисел i переход будет $dp_{i,\ left\_prev,\ left\_curr} += dp_{i,\ left\_prev,\ left\_curr+3}$
- Для троек подряд идущих чисел нас будет интересовать только вариант $(i-1, i, i+1)$, так как мы хотим потратить все оставшиеся числа $i-1$ и можем сделать это единственным способом — использовать их для троек подряд идущих (по инварианту динамики). Переход для таких троек будет выглядеть как $dp_{i+1,\ left\_curr-left\_prev,\ cnt_{i+1}-left\_prev} += dp_{i,\ left\_prev,\ left\_curr}$ — мы потратили ровно $left\_prev$ чисел для троек вида $(i-1,\ i,\ i+1)$.

Для корректного пересчета первый вид троек стоит учесть до того, как считать следующий слой для второго вида троек, так как мы хотим поддерживать инвариант, что все тройки вида $(i,\ i,\ i)$ уже использованы при пересчета для слоя $i+1$. Ответ на задачу будет лежать в $dp_{m,0,0}$. Заметим, что $left_prev,\ left_curr \leqslant n$ для каждого $i : 1 \leqslant i \leqslant m$, что означает, что решение работает за $O(m \cdot n^2)$. Памяти решение использует столько же, что не хватает для прохождения лимита по памяти. Память можно оптимизировать, заметив, что для пересчета динамики нам нужно сохранять только слои i и $i+1$, а все ненужные слои можно переиспользовать, таким образом вместо $O(m \cdot n^2)$ памяти получим $O(n^2)$.

Подзадача 6

Для решения финальной подзадачи необходимо слегка оптимизировать динамику из подзадачи 5. Для этого нужно перебирать $left\_prev$ и $left\_curr$ в границах от 0 до соответствующего им cnt. Теперь решение для всех m суммарно будет работать за $\sum_{i=1}^{m} cnt_{i-1} \cdot cnt_i$. Нетрудно заметить, что такую сумму можно ограничить сверху n^2, зная, что $cnt_1 + cnt_2 + \dots cnt_m = n$: $n^2 = (cnt_1 + cnt_2 + \dots cnt_m) \cdot (cnt_1 + cnt_2 + \dots cnt_m) = cnt_1 \cdot (cnt_1 + cnt_2 + \dots cnt_m) + cnt_2 \cdot (cnt_1 + cnt_2 + \dots cnt_m) + \dots + cnt_n \cdot (cnt_1 + cnt_2 + \dots cnt_m)$, что очевидно не меньше, чем $\sum_{i=1}^{m} cnt_{i-1} \cdot cnt_i$, так как $cnt_i \geqslant 0$. Получаем итоговую асимптотику $O(m + n^2)$.

Задача 8. Обходы бинарного дерева

Подзадача 1

Чтобы найти текущий обход, реализуем алгоритм, описанный в условии. Это поиск в глубину, в котором порядок обхода детей определяется числом, записанным в вершине. На каждый запрос второго типа явно выпишем обход и найдём в нём соответствующую вершину. Это решение работает за $O(nq)$.

Подзадача 2

Заметим, что обход меняется q_1 раз, и, если q_1 невелико, то можно явно найти все эти обходы за $O(nq_1)$. Чтобы отвечать на запросы второго типа, вместе с обходом найдём позиции всех вершин в этом обходе, то есть просто обратную перестановку. Таким образом, мы сможем отвечать на запросы второго типа за $O(1)$. Суммарно это решение работает за $O(nq_1 + q)$.

Подзадача 3

Эта задача похожа на предыдущую, поскольку необходимо отвечать на запросы второго типа только для одного обхода. Чтобы явно найти этот обход, найдём числа, написанные на всех вершинах, после всех запросов первого типа. Это можно сделать с помощью метода сканирующей прямой. Отсортируем все отрезки, соответствующие запросам, по возрастанию правой границы и пройдёмся по массиву, поддерживая множество текущих отрезков в очереди с приоритетом или другой подобной структуре. Число, записанное на текущей вершине, равно новому значению в самом позднем из покрывающих её запросов. Эта часть решения работает за $O((n+q)\log q)$. Остаётся найти итоговый обход и позиции всех вершин в нём, после чего мы сможем отвечать на запросы второго типа за $O(1)$. Суммарно это решение работает за $O((n + q)\log q)$.

Подзадача 4

Рассмотрим прямой (то есть изначальный) обход нашего дерева.

Рассмотрим изменение позиции вершины i после запроса первого типа. Заметим, что изменения чисел на вершинах, которые не лежат на пути из i до корня, не влияют на ответ.

От числа в вершине i зависит её позиция в обходе относительно вершин в её поддереве. Если число в i стало равно 1, то её позиция увеличится на сумму размеров её левого и правого поддерева. Если число в i стало равно 0, то его позиция увеличится на размер её левого поддерева.

Если число в предке i стало равно 1, то теперь этот предок будет записан после своих поддеревьев; таким образом, позиции всех его потомков уменьшатся на 1. То есть позиция i уменьшится на 1. Если число в предке i стало равно 0, то теперь позиции всех вершин в его левом поддереве уменьшатся на 1. То есть позиция i уменьшится на 1, если i лежит в левом поддереве этого предка.

Отсюда, чтобы ответить на запрос второго типа, нам необходимо знать число, записанное на вершине i, количество предков i, на которых записано число 1, и количество предков i, на которых записано число 0, таких, что i лежит в левом поддереве этого предка.

Ключевое наблюдение в этой подгруппе заключается в том, что высота такого дерева равна $O(\log n)$. Действительно, в таком дереве высоты h $2^{h-1}-1$ вершин. Таким образом, в запросе второго типа можно просто перебрать всех предков вершины.

Чтобы поддерживать числа на вершинах, можно использовать дерево отрезков с массовыми операциями, которое должно поддерживать присваивание на отрезке и запрос в точке. Используя такую структуру, запрос первого типа можно обрабатывать за $O(\log n)$.

Предподсчитать размеры поддеревьев можно за $O(n)$ с помощью динамического программирования по поддеревьям. Таким образом, узнать, на сколько увеличится позиция вершины i, можно за $O(1)$.

Чтобы обработать запрос второго типа, пройдёмся по предкам вершины i и для каждого из них узнаем записанное на нём число. Отсюда мы узнаем, на сколько уменьшится позиция вершины i. Таким образом, мы сможем обработать запрос второго типа за $O(\log^2 n)$.

Суммарно это решение работает за $O(n + q \log^2 n)$.

Подзадача 5

Первый способ решения этой подзадачи использует наблюдения, сделанные в подзадаче 4. Действительно, если мы меняем число в вершине на 1, то позиции всех вершин в её поддереве уменьшаются на 1. Аналогично, если мы меняем число в вершине на 1, то позиции всех вершин в её левом поддереве уменьшаются на 1.

Заметим, что, если выписать прямой обход дерева, то левое и правое поддеревья вершины будут образовывать непрерывные отрезки. Таким образом, если построить дерево отрезков с массовыми операциями, которое поддерживает прибавление на отрезке и запрос в точке, можно обработать запрос первого типа или узнать всё необходимое для ответа на запрос второго типа за $O(\log n)$. Это решение работает за $O(n + q \log n)$.

Второй способ решения этой подзадачи следует непосредственно из определения обхода: если мы меняем число в вершине, то в обходе она переставляется на позицию до её поддеревьев, между её поддеревьями или после её поддеревьев соответственно. Таким образом, обход можно поддерживать в дереве поиска, например, декартовом дереве, которое поддерживает операции split и merge.

Чтобы отвечать на запросы второго типа, сохраним указатели на вершины дерева поиска, соответствующие вершинам в обходе. Будем поддерживать размеры поддеревьев и родителя вершины в дереве поиска. Таким образом, чтобы ответить на запрос второго типа, необходимо подняться до корня в дереве поиска и найти номер вершины в обходе. Это решение также работает за $O(n+q \log n)$.

Подзадача 6

В этой подгруппе можно представить его как массив, а запросы на пути до корня — как запросы на префиксе.

Воспользуемся методом корневой декомпозиции. Разобьём вершины на $\lceil \frac{n}{B} \rceil$ блоков: с 1 по B, с $B+1$ по $2B$ и так далее. Внутри каждого блока отсортируем вершины в порядке возрастания глубины. Будем поддерживать количество вершин на префиксе блока, на которых записано число 1.

Рассмотрим запрос на присваивание числа x на отрезке $[l, r]$. Этот отрезок покрывает $O(\frac{n}{B})$ блоков целиком и $O(1)$ блоков частично. Если блок покрыт целиком, то префсуммы на нём считаются тривиально (так как теперь все числа на вершинах равны x), и мы можем обработать этот случай за $O(1)$. Если блок покрыт частично, пересчитаем его полностью. Таким образом, мы обработали операцию за $O(B + \frac{n}{B})$.

Для запроса второго типа необходимо найти количество вершин на пути до корня, на которых записано число 1. Заметим, что в этот путь входит префикс вершин каждого блока. Таким образом, если мы знаем размеры этих префиксов, мы можем получить ответ на запрос за $O(\frac{n}{B})$. Получать размеры можно бинпоиском, но есть более эффективное решение.

Так как мы знаем все запросы заранее, мы можем отсортировать их и предпосчитать для каждого блока соответствующие размеры префиксов двумя указателями за $O(q \log q + (q+B)B)$. Этот метод

требует $O(qB)$ памяти на сохранение предподсчёта. Однако мы можем избавиться от этого, если для каждого блока пройдёмся по запросам отдельно.

Итого получаем асимптотику $O((n+q)\sqrt{n})$ с $O(n+q)$ памяти при $B \approx \sqrt{n}$.

Подзадача 7

Обобщим решение подгруппы 6 на произвольное бинарное дерево.

Разобьём вершины на блоки по B. Построим для каждого блока сжатое дерево, то есть дерево на вершинах блока, такое что p — родитель i, если p — самая глубокий предок i, который содержится в блоке. (Для блоков, не содержащих корень исходного дерева, может быть полезно ввести фиктивный корень.) Вместо префсумм будем поддерживать суммы на пути до корня для каждой вершины.

Легко видеть, что запросы обрабатываются аналогично предыдущему решению. Для предпосчёта запросов к фиксированному блоку вместо двух указателей можно обойти дерево в глубину, поддерживая самую глубокую вершину из блока, являющуюся предком текущей.

Подзадача 8

Найдём для каждой вершины i вершину L_i — самого глубокого предка i такого, что вершина лежит в его левом поддереве. Это можно сделать за $O(n)$, используя поиск в глубину. Заметим, что, если вершина i лежит в левом поддереве её предка p, то $p = L(L(\dots L(i)\dots))$.

Если соединить ребром вершины i и L_i, получится набор деревьев, или лес L. Из предыдущего свойства следует, что предки вершины i, в левых поддеревьях которых она лежит — это предки i в L.

Поскольку в этой подзадаче у каждой вершины не более одного предка, L представляет собой набор изолированных вершин (тех, для которых не существует L_i) и бамбук (дерево, в котором у каждой вершины не более одного ребёнка) с подвешенными к некоторым его вершинам дополнительными вершинами (с равными L_i). Таким образом, обрабатывать запросы на L можно так же, как в подзадаче 6 (поддерживая количество вершин, на которых записано число 0).

Подзадача 9

Аналогично решению предыдущей подзадачи, построим лес L. Всё, что остаётся сделать — это применить на нём идею, описанную в подзадаче 7.

Решения подзадач 7, 8 и 9 также работают за $O((n+q)\sqrt{n})$.