Разбор задачи «Ослабление флота»

Решение на **47 баллов**: Напишем ровно то, что просят в условии. n раз будем искать медиану в массиве, после чего удалять ее, получая новый массив. Как найти медиану? Есть два способа:

- Отсортировать массив за $O(n \log n)$, тогда медианой, очевидно, будет $\frac{n}{2}$-й элемент. Итоговая сложность решения будет $O(n^2 \log n)$.
- Воспользоваться алгоритмом поиска к-ой порядковой статистики, который работает за $O(n)$, а также реализован в стандартной библиотеке в некоторых языках. Итоговая сложность решения будет $O(n^2)$.

Для решения на **100 баллов** заметим, что можно модифицировать первый способ решения предыдущей подгруппы. После того, как мы отсортировали массив и удалим медиану, следующая медиана это соседний элемент от удаленного. Таким образом, достаточно отсортировать массив ровно один раз, после чего с помощью метода двух указателей пройтись по нему и вывести нужные элементы.

Для удобства можно отдельно рассмотреть случай нечетной длины. Итоговая сложность данного решения $O(n \log n)$.

Разбор задачи «Рапорт»

Решение на **51 балл**: Обе части рапорта можно написать, если ширина левой части рулона не меньше максимальной длины слова в первой части рапорта, и, аналогично, ширина правой части рулона не меньше максимальной длины слова во второй части рапорта. Переберем положение вертикальной черты, при котором рапорт возможно написать. После этого, проэмулируем процесс написания слов и вычислим длину рулона для данной позиции вертикальной черты. Ответом является минимальная длина рулона по всем положениям черты.

Решение на **100 баллов**: Заметим, что при сдвигании вертикальной черты вправо (то есть, при увеличении ширины левой части рулона и уменьшении ширины правой части рулона), длина первой части рапорта будет уменьшаться, а длина второй части рапорта будет увеличиваться (длиной части рапорта назовем минимальную длину рулона, необходимую, чтобы написать эту часть рапорта на соответствующей части рулона).

Поэтому, используя двоичный поиск можно найти две соседних вертикальных черты, что если провести одну из них, левая часть будет длиннее правой, а если вторую, то наоборот. Заметим, что среди ответов для этих двух черт обязательно есть оптимальный ответ.

Для того, чтобы найти длины частей для определенной черты, нужно просто проэмулировать процесс, добавляя по одному слову.

Итоговая асимптотика: $O((n+m)\log w)$.

Разбор задачи «Декодирование сообщения»

Общее замечание: так как каждый символ исходной строки кодируется в трехзначное число, строку s можно разбить на последовательные блоки из 3 цифр, каждый из которых будет соответствовать закодированной букве.

Решение на **21** балл: Так как длина искомой исходной строки не превосходит $\frac{12}{3} = 4$, достаточно перебрать все возможные исходные строки (их не больше $(26+26)^4 = 52^4 < 10^7$. Для каждой строки останется только проверить, что с помощью $d = 1$ ее можно закодировать в строку s.

Решение на **43** балла: Заметим, что разные блоки из 3 цифр в строке s независимы, таким образом, достаточно для каждого блока b найти все возможные символы c, которые при $d = 1$ можно закодировать в b, а после этого перемножить эти количества для всех блоков.

Решение на **75-100** баллов: Заметим, что каким бы ни было число d, из двузначного кода символа нельзя получить число в диапазоне $[100, 122]$ (соответствующего диапазону символов [«d»..«z»]. Соответственно, эти символы декодируются однозначно.

Теперь переберем число d, а затем для каждого блока из 3 цифр в s найдем все возможные символы исходной строки, которые могут соответствовать этому блоку (это можно сделать за $O(1)$, достаточно попробовать удалить из блока каждую из трех цифр, а затем проверить, что оставшееся

число находится либо в отрезке [65, 90], либо в отрезке [97, 99]). Все эти количества для фиксированного d опять же перемножим между собой и добавим к ответу.

В зависимости от оптимальности реализации этого алгоритма можно было получить либо 75, либо 100 баллов.

Разбор задачи «Кот Гусь и случайная матрица»

Чтобы решить **первую подзадачу**, достаточно реализовать любое решение за $O((nm)^2)$, например, перебрать координаты двух углов, и посчитать с помощью частичных сумм сумму на подматрице.

Чтобы решить **вторую подзадачу**, необходимо уже реализовать решение за $O(n^2m)$, перебрав первую и последнюю строку подматрицы, и, таким образом, свести подзадачу к подзадаче с $n = 1$.

Чтобы окончательно решить вторую подзадачу, необходимо придумать, как решить **третью подзадачу**. На одной прямой (вертикальной или горизонтальной) необходимо найти подотрезок с максимальной суммой, делящейся на p. Это можно сделать, посчитав префиксные суммы, для каждого остатка по модулю p, а затем рассмотреть наибольшую и наименьшую суммы с таким остатком.

Таким образом, первые три подзадачи возможно решить без использования случайности входных данных.

Чтобы решить **четвертую подзадачу**, нужно воспользоваться случайностью входных данных, и реализовать более медленно правильное решение, не будем на этом останавливаться и сразу перейдем к решению, набирающему 100 баллов.

Будем строить подматрицы в порядке убывания суммы, от большей к меньшей. Для этого изначально возьмем всю матрицу полностью. Очевидно, она является подматрицой с наибольшей суммой. Какой может быть вторая по сумме матрица? Конечно, это вся матрица, из которой удалили какую-либо крайнюю строку/столбец. Аналогично, можно развить эту идею, поддерживая подматрицы в set. Каждый раз, можем доставать матрицу, которая является максимальной на данный момент, и обрезать у нее по одному строке и столбцу. Как только мы нашли матрицу с суммой делящейся на p, стоит вывести ее как ответ.

Если реализовать это с помощью set, то решение может набирать **80-100 баллов**.

Чтобы получить **100 баллов**, нужно реализовать это с помощь priority queue, не используя проверку, что такую матрицу уже рассматривали, а поддерживая указатель ptr на текущую границу, которую требуется отрезать, и сначала увеличивать x_1, затем, возможно, сдвигая указатель, затем уменьшать x_2, если указатель сейчас стоит на этой координате, и так далее.

Таким образом путь до каждой матрицы будет ровно один — последовательно подставляющий нужные координаты по очереди. В итоге, в priority queue будет храниться $(x_1, x_2, y_1, y_2, ptr)$, где ptr принимает значения 0..3.

Можно доказать, что с очень высокой вероятностью, количество матриц, которые придется рассмотреть, достаточно мало, а именно, порядка $O(p)$.

Также возможны решения, которые перебирают матрицы только «достаточно» большого размера.

Разбор задачи «Упорядочивания»

В задаче нужно было посчитать количество топологических сортировок дерева, ребра которого ориентированы. Напомним, что топологической сортировкой графа называется такое упорядочивание вершин, при котором любое ребро ведет из вершины с меньшим номером в вершину с большим номером.

Решение на **16 баллов**. Переберем все возможные перестановки чисел от 1 до n. Для каждой из них проверим, удовлетворяет ли она необходимым условиям. Итоговая сложность решения — $\mathcal{O}(n! \cdot n)$.

Решение на **32 балла**. Воспользуемся динамическим программированием по подмножествам. Пусть $dp[S]$ — количество перестановок вершин из множества S, которые удовлетворяют необходимым условиям, где S — подмножество множества чисел от 1 до n. Чтобы пересчитывать динамику, переберем, какую вершину мы добавляем в множество, и если не существует ребер из новой вершины в текущее множество, ее можно добавить и обновить значение динамики.

Решения за полиномиальное время. Чтобы получить решение, работающее за полиномиальное время, подвесим дерево за произвольную вершину, то есть выберем одну вершину и сделаем ее корнем дерева, а для всех остальных вершин определим их родителя. Пусть $sz[u]$ — размер поддерева вершины u. Эту величину мы можем посчитать заранее обходом в глубину.

Решение для частного случая: все ребра вниз. Решим сначала более простой частный случай задачи: пусть все ребра направлены вниз, то есть от родителя к ребенку. В этом случае задачу можно решить с помощью динамического программирования по поддеревьям. Пусть $dp[u]$ — количество топологических сортировок поддерева вершины u. Пусть вершина u имеет k детей: $v_1, v_2, \ldots, v_k$. Рассмотрим какую-нибудь топологическую сортировку поддерева u. На первом месте в ней всегда обязана стоять вершина u, так как все ребра в ее поддереве направлены вниз. После этого у нас останется $sz[u] - 1$ свободных позиций, которые займут вершины из поддерева u, кроме самой u. Заметим, что мы можем произвольным образом выбрать подмножество позиций, которые займут вершины из поддерева v_1, затем произвольным образом выбрать подмножество позиций, которые займут вершины из поддерева v_2, и так далее по всем сыновьям u. После этого мы можем каждое из поддеревьев вершин $v_1, \ldots, v_k$ топологически отсортировать. Несложно убедиться, что после этого мы всегда получим какую-то топологическую сортировку поддерева u, а также любая топологическая сортировка поддерева u может быть получена таким способом.

Итак, сначала нам нужно из $sz[u] - 1$ свободных позиций выбрать $sz[v_1]$ позиций для вершин поддерева v_1. После этого нужно из $sz[u] - 1 - sz[v_1]$ свободных позиций выбрать $sz[v_2]$ позиций для вершин поддерева v_2, и так далее. На i-м шаге у нас есть $sz[u] - 1 - sz[v_1] - \ldots - sz[v_{i-1}]$ свободных позиций, из которых нужно выбрать $sz[v_i]$ для вершин поддерева v_i. Количество способов сделать это — это $C^{sz[v_i]}_{sz[u]-1-sz[v_1]-\ldots-sz[v_{i-1}]}$. Здесь через C^k_n мы обозначаем количество сочетаний из n по k. Эти значения можно посчитать заранее с помощью треугольника Паскаля.

Таким образом, общее количество способов распределить места составляет

$$C^{sz[v_1]}_{sz[u]-1} \cdot C^{sz[v_2]}_{sz[u]-1-sz[v_1]} \cdot C^{sz[v_3]}_{sz[u]-1-sz[v_1]-sz[v_2]} \cdot \ldots \cdot C^{sz[v_k]}_{sz[u]-1-sz[v_1]-\ldots-sz[v_{k-1}]}$$

Полученный результат нужно также умножить на все значения $dp[v_1], dp[v_2], \ldots, dp[v_k]$, так как после того, как мы распределили места, мы можем произвольным образом выбрать топологическую сортировку для каждого из поддеревьев $v_1, \ldots, v_k$.

Посчитав таким образом динамику для всех поддеревьев исходного дерева, мы получим ответ для случая, когда все ребра ведут вниз.

Полное решение. Пусть теперь некоторые ребра ведут наверх. Удалим из дерева все ребра, ведущие наверх, и решим задачу для оставшихся ребер. Для этого внутри каждого получившегося дерева посчитаем динамику, описанную выше, а после этого скомбинируем ответы: чтобы получить топологическую сортировку всего графа, нужно произвольным образом выбрать для каждого из отдельных деревьев множество позиций, которые будут занимать вершины из этого дерева, а также топологическую сортировку внутри вершин этого дерева. Формула, по которой можно получить ответ, идентична той, с помощью которой мы считали динамику для первого случая.

Заметим, что не любое упорядочивание вершин среди тех, которые мы посчитали, подходит. Некоторые из полученных упорядочиваний нарушают условие для каких-то ребер, направленных вверх. Пусть есть s ребер, направленных вверх. Пронумеруем их произвольным образом от 1 до s. Пусть A_i — множество упорядочиваний вершин, в которых конца ребра вверх под номером i расположены в неправильном порядке. Тогда, чтобы получить правильный ответ, нам нужно посчитать размер множества $A_1 \cup A_2 \cup \ldots \cup A_s$ и вычесть его из ответа, который мы уже посчитали. Воспользуемся для этого формулой включений-исключений.

Пусть для всех подмножеств ребер, ведущих наверх, мы посчитали количество топологических сортировок, в которых для каждого из этих ребер (и, быть может, для каких-то других) условие нарушено. Тогда чтобы посчитать размер объединения множеств A_i, нужно сложить эти значения для подмножеств с нечетным числом ребер и вычесть значения для подмножеств с четным числом ребер.

Однако перебор всех возможных подмножеств ребер, ведущих наверх, занимает экспоненциальная время. Чтобы не делать этого, добавим несколько необходимых нам параметров динамики. Дополнительно нам понадобится знать, какова четность количества ребер вверх, для которых условие

заведомо нарушено, а также размер текущего поддерева текущей вершины, если учитывать только ребра вниз. Иными словами, нам нужно хранить в состоянии динамики размер текущего поддерева вершины с учетом того, что некоторые ребра вверх мы удалили, а некоторые развернули в другую сторону. Пусть теперь $dp[u][size][parity]$ — это количество упорядочиваний поддерева вершины u, в которых текущее поддерево вершины u по ребрам вниз имеет размер $size$, а число ребер вверх, которые мы перевернули, дает остаток $parity$ от деления на 2. Будем пересчитывать эту динамику, добавляя детей очередной вершины по очереди. Каждый раз, когда мы встречаем ребро дерева, ведущее вниз, мы можем только добавить к текущему поддереву по ребрам вниз новую часть. Если же мы встречаем ребро вверх, то мы можем либо убрать его, либо развернуть, изменив при этом четность числа ребер, порядок на которых нарушен. Каждый раз, когда мы выкидываем очередное ребро вверх, будем домножать ответ на соответствующее число сочетаний. В конечном итоге в значениях динамики в корне дерева мы получим необходимые слагаемые для формулы включений-исключений.