Задача A. Ода ветру

Пусть s — это сумма элементов массива a. Если s — составное, то a уже является наибольшим подмножеством с составной суммой. В противном случае, так как $n \geqslant 3$, s должно быть простым числом большим двух 2, то есть s должно быть нечетным. Далее заметим, что так как все элементы a различны, то при удалении любого числа из a оставшаяся сумма все еще будет строго больше 2. А это наталкивает на следующее решение: если s — простое, удаление любого нечетного числа из a даст нам подмножество с составной суммой размера $n-1$. Все потому, что при нечетном s в самом массиве a должно присутствовать хотя бы одно нечетное число, а разность двух нечетных всегда четно. А так как мы утверждаем, что данная разность получится не менее 4, то новая сумма будет всегда составной.

Задача B. Омкар и божественное дерево

Так как количество ограничений строго меньше n, то обязательно найдется хотя бы одно число от 1 по n, которое не является значением b ни для какого из ограничений. Найдем такую вершину (которая не является значением b ни для какого ограничения) и построим дерево, являющимся графом-«звездой» с центром в данной вершине. Простой способ построения — это добавить ребра из данной вершины во все остальные вершины от 1 по n.

Задача C. Омкар и определение

Для начала заметим, что в определяемой таблице невозможна ситуация, при которой для какой-то клетки и клетка сверху и клетка слева заполнены. Если бы такая ситуация присутствовала, то данная клетка не будет являться хорошей, независимо от того, свободна она или заполнена, а потому мы не смогли бы однозначно определить ее состояние.

Далее заметим, что в любой таблице с описанным выше свойством, вы буквально из каждой клетки можете двигаться либо в свободную клетку вверх, либо в свободную клетку влево (или в обе стороны), а потому, каждая свободная клетка обязана быть хорошей — просто двигайтесь вверх или влево (куда можно), пока не покинете таблицу.

Из этого следует, что для любой таблицы с заданным свойством, имея на руках только хорошие клетки и начиная с внешних клеток вы сможете определить, что нехорошие клетки заполнены. Из чего следует, что клетки далее удовлетворяют описанному свойству, и снова нехорошие клетки являются заполненными, и так далее. Данные размышления в результате позволяют определить все клетки таблицы (как так хорошие клетки, очевидно, свободные).

В итоге определяемость таблицы эквивалентно проверке, что каждая из ее клеток имеет свободную клетку непосредственно сверху и/или слева. Вы можете проверить данное свойство для произвольных подматриц, предпосчитав двухмерные префиксные суммы количества клеток, в которых данное правило нарушается, а потом просто проверяя, что сумма на подматрице равна 0. Данное решение работает за $O(nm+q)$.

Текущая версия задачи спрашивает только про подматрицы, которые содержат все строки, что сделано для упрощения реализации.

Задача D. Омкар и смысл жизни

Решение 1

Определим для каждого j, индекс next_j такой, что $p_{\text{next}_j} = p_j + 1$.

Для каждого индекса j, сделаем запрос со всеми 1-ми кроме $a_j = 2$. Если ответ k существует, то мы можем понять, что $\text{next}_k = j$.

Также, для каждого j, сделаем запрос со всеми 2-ми кроме $a_j = 1$. Если ответ k существует, то мы можем понять, что $\text{next}_j = k$.

Для каждого j такого, что $p_j \neq n$, либо $\text{next}_j > j$, и в таком случае в первой группе операций мы определим next_j, либо $\text{next}_j < j$, а в этом случае во второй группе операций мы определим next_j. А потому мы полностью определим весь массив next.

Чтобы посчитать p, заметим, что индекс j с $p_j = 1$ не появится в массиве next. А потому найдем такой j и присвоим $p_j = 1$. Далее присвоим j значение next_j, а $p_j = 2$, и так далее.

Суммарное количество использованных операций равно $2n$, то есть строго сколько можно.

Решение 2

Для начала определим $q_j = p_j - p_n$ для всех j.

Для каждого значения x от $-(n-1)$ по $n-1$ (это единственные возможные значения $p_j - p_n$), если x — неотрицательное, то сделаем запрос, в котором все $a+1$ равны x кроме $a_n = 1$; иначе сделаем запрос, в котором все a равны 1 кроме $a_n = 1 - x$. Если ответ k существует, то $q_k = x$.

Заметим, что существует не более одного k, для которого $q_k = x$ для каждого x, А потому мы полностью определим массив q таким способом (очевидно, что мы сами проставим $q_n = 0$).

p_n в таком случае равно количеству j, что $q_j \leqslant 0$. А зная это мы далее определяем все оставшиеся p как $p_j = q_j + p_n$.

Суммарное количество операций равно $2n - 1$, что на 1 меньше лимита.

Бонусный вопрос: Оптимизируйте данное решение до n запросов.

Задача E. Момент цветения

Пусть f_v — это количество раз, сколько v появляется в q запросах. Если f_v — нечетное для какого-то $1 \leqslant v \leqslant n$, то не существует способа выбрать пути, делающие у всех ребер четные веса. Чтобы понять почему, заметим, что один запрос соответствует ровно одному ребру, инцидентному v. Если нечетное количество путей инцидентно v, то как минимум одно ребро, инцидентное v, будет иметь нечетный вес.

Оказывается, что это единственное свойство, которое нужно проверить. Другими словами, если f_v — четное для всех v, то существует способ выбора путей, делающих у всех ребер четные веса.

Предположим, что все f_v — четные. Тогда мы можем сделать следующее: возьмем любой остов графа и выберем для каждого пути путь от a до b в этом дереве.

Интуитивно понимание следующее. Предположим, что остов — это цепь. Тогда каждый запрос превращается в отрезок и мы проверяем, что все точки в данном отрезке покрыты четное количество раз. Для того чтобы каждой точке быть покрытой четное количество раз, нам нужно, чтобы каждая точка присутствовала в четном количестве запросов. Обобщая на дерево: когда первый путь из a_1 в b_1 прибавлен, чтобы сделать их снова четными, нам нужны пути, которые перекроют отрезок из a_1 в b_1. Один из способов — это использовать какие-то два пути из a_1 в c и из c в b_1. Заметим, что если новый отрезок, которые делает путь из a_1 в b_1 четным испортит какие-то другие ребра, позже они тоже будут исправлены.

Задача F. Защитник детских мечт

Минимальное количество цветов, которые необходимы, равно $\lceil \log_k n \rceil$.

Для достижения данной оценки, вы можете разделить все вершины в k последовательных отрезков равной длины (или как можно равнее). Любое ребро между вершинами в разных подотрезках, вы красите в 1, например. Далее, вы рекурсивно раскрашиваете данные отрезки, исключив уже использованный цвет.

Любой путь одного цвета всегда находится между подотрезками одного размера внутри одного большего подотрезка (или всего массива). Так как всего только k таких отрезков, то путь будет длины не более $k - 1$.

Максимальная глубина рекурсии равна $\lceil \log_k n \rceil$, что является желаемым количеством цветов.

Далее докажем, что $\lceil \log_k n \rceil$ обязательно необходимо. Для этого докажем эквивалентное утверждение, что при валидной раскраске из c цветов, n не может превосходить k^c. Это утверждение докажем по индукции по c.

База индукции: $c = 0$. Если у вас нет цветов, то вы не можете покрасить ни одно ребро, а потому n не может превосходить $1 = k^0$.

Для шага индукции предположим, что граф с любой валидной раскраской, использующей не более $c-1$ цветов, может иметь не более k^{c-1} вершин. Мы хотим показать, что граф с любой валидной раскраской в не более c цветов содержит не более k^c вершин. Для этого выберем произвольный цвет и разобьем все вершины в не более чем k групп так, что внутри каждой группы нет ребер выбранного цвета. Следовательно, все группы покрашены в не более чем $c - 1$ цветов, а потому имеют размер не более k^{c-1} вершин. В сумме получаем не более $k \cdot k^{c-1} = k^c$ вершин.

Разбиение производим следующим образом: мы разобьем вершины на множества $s_0, s_1, \ldots, s_{k-1}$, где s_j содержит все вершины a, такие что длина наидлиннейшего пути, оканчивающегося в a, который состоит только из ребер выбранного цвета, строго равен j. Данная длина не превосходит $k-1$, так как в нашей раскраске нет путей длины k одного цвета. Более того, не может существовать ребер выбранного цвета внутри одного множества s_j, иначе бы длина наидлиннейшего пути в конец такого ребра была бы на единицу больше, чем в начало данного ребра, то есть $j+1$.

Таким образом, граф с любой валидной раскраской в c цветов не может иметь более c^k вершин, то есть нам необходимо использовать хотя бы $\lceil \log_k n \rceil$ в нашей конструкции, что и было показано ранее.

Задача G. Омкар и путешествия во времени

После каждого перемещения во времени Окабэ получает новое множество выполненных заданий. Примем данное утверждение бездоказательно на данный момент, но далее это будет доказано в ходе общего решения.

Таким образом, нам необходимо посчитать количество различных множеств, которые получатся перед первым из тех, что содержат s как подмножество. Но для начала нужно понять, какие множества в принципе могут появиться в виде множества выполненных заданий; назовем такие множества валидными.

Представим задания как отрезки $[a_k, b_k]$.

Для начала, заметим, что не все множества валидны. Если у вас есть интервалы $[1,2]$ и $[3,4]$, очевидно, что множество $\{[3,4]\}$ не валидно. Более того, можно заметить, что правило применимо и к паре отрезков $[1,3]$ и $[2,4]$ ($\{[2,4]\}$ — не валидное множество).

Данное правило обобщается следующим образом: если есть два отрезка $[a,b]$ и $[c,d]$, где $a < c$ и $b < d$, то любой валидный сет, содержащий $[c,d]$, должен также содержать и $[a,b]$. Все потому, что если был достигнут момент d для выполнения задания $[c,d]$, то момент b уже автоматически был достигнут и задание $[a,b]$ было выполнено (так как $b < d$). С другой стороны, любое перемещение, которое откатывает задание $[a,b]$ также откатывает и задание $[c,d]$ (так как $a < c$).

Оказывается, что правило выше является необходимым и достаточным для того, чтобы множество было валидным; оно, очевидно, необходимое и далее мы докажем, почему оно также является достаточным, но у вас уже может быть интуитивное ощущение, почему это правда.

Для решения задачи нам необходимо понять, как по двум заданным множествам понять, какое из них Окабэ получит сначала. Сначала, для любой пары валидных множеств, посмотрим на их последние отрезки (т. е. на отрезки с максимальным значением b). Если они различаются, что множество с отрезком с большим значением b встретится позже.

Все потому, что для любого валидного множества, максимальное значение b среди всех отрезков равно максимальному моменту времени b, которого Окабэ когда-либо достигал. Это можно понять заметив, что единственный способ отменить данное задание — это выполнить задание с большим значением b; любое задание с меньшим b либо содержится (как отрезок) в данном задании, и в таком случае не может его отменить, либо у него и значение a меньше, а в таком случае мы доказали, что валидных множествах оно уже должно быть выполнено.

Так как максимальное значение b, которое когда-либо достигал Окабэ, будет становиться только больше, то соответственно и множества с большими максимальными значениями b будут встречаться позже.

Более того, мы можем видеть, что для любых двух валидных множеств с одинаковыми максимальными b, можно отбросить этот отрезок и сравнивать по второму максимуму.

Данный подход дает нам порядок валидных множеств. Можно доказать, что описанного ранее свойства достаточно для доказательства критерия валидности множества, показав, что для каждого валидного множества следующим встреченным множеством будет именно то, что идет в отсортированном порядке. Детали доказательства оставим читателям.

Для того чтобы наконец решить задачу, удобно представлять валидные множества другим способом. А именно, мы можем представить валидное множество v как множество u отрезков, нахождение которых в множестве не является следствием нахождения в множестве v каких-то других отрезков из v. Думая о свойстве выше, можно увидеть, что u — это, в действительности, множество отрез-

ков, рекурсивно содержащихся друг в друге; т. е. оно содержит отрезок, который содержит другой отрезок, который содержит другой отрезок и так далее.

Будем считать представление выше упорядоченным так, что последним будет отрезок, содержащий все остальные, а первым — тот, что содержится во всех остальных.

Теперь мы можем решать задачу. Для заданного множества s, сначала определим его представление; это легко сделать с помощью сортировки и выкидыванием ненужных отрезков. Валидные множества, также в соответствующем представлении, которые встретятся перед s, а потому меньшие, если игнорировать их общий суффикс с s, то имеют значение b последнего отрезка меньшее, чем b последнего отрезка s.

А потому мы можем решать задачу следующим образом. Будем подсчитывать количество описанных выше множеств для каждой длины общего суффикса. Для каждого суффикса, пусть последний отрезок в s, который не входит в общий суффикс, равен x, и пусть первый отрезок в суффиксе равен y. Количество множеств для такого суффикса равно количество количеству рекурсивных множеств, у которых максимальный отрезок содержится в y и значение b которого меньше значения b для x.

Мы можем считать данное значение так. Будем поддерживать структуру данных с суммой на отрезке как binary index tree. Эта структура будет хранить для каждого a, количество рекурсивных подмножеств, у которых максимальный отрезок с данным значением a (после того, как этот отрезок был обработан). Будем обрабатывать эти отрезки в порядке возрастания b.

Для каждого отрезка x, чтобы положить его в структуру данных, мы можем просто сделать запрос на сумму прямо по отрезку x, и добавить 1 к результату. Что будет равно количеству рекурсивных множеств с максимальным отрезком x, а потому мы просто вставляем получившиеся значение в структуру в значение a x-а.

Перед тем, как вставить x в структуру, если оно в представлении s, то мы можем определить ответ для суффикса s, который содержит все отрезки правее x следующим образом. Так как отрезки, находящиеся в текущий момент в структуре данных, — это в точности отрезки со значением b меньше чем x, ответ для суффикса — это просто запрос на сумму на отрезке y, где y — это прямо следующий отрезок после x в s. А потому мы делаем запрос на отрезке и добавляем его к ответу.

Заметим, что все эти подсчеты не учитывают само s, но учитывают пустое множество, которое учитывать не нужно, а потому ответ сходится.

Асимптотика решения равна $O(n \lg n)$.

Задача H. Омкар и туры

Для начала заметим, что мы обрабатываем запросы оффлайн. Мы можем отсортировать запросы по количеству автомобилей и обрабатывать их в порядке убывания.

Теперь, рассмотрим версию задачи, если бы все значения удовольствия были бы различны. Тогда всегда бы существовал ровно один достижимый город с максимальным уровнем удовольствия. Чтобы решать такую задачу, мы можем поддерживать DSU и хранить в нем для каждой компоненты связанности максимальный уровень удовольствия и номер вершины с данным уровнем, которые обозначим за $enj[u], mxi[u]$ для компоненты u. При слиянии двух компонент связанности u, v, мы просто пересчитываем $enj[u] = \max(enj[u], enj[v]), mxi[u] = \arg\max(mxi[u], mxi[v])$. Теперь, рассмотрим обработку запроса со стартовой вершиной a и количеством автомобилей x, назовем ее «компоненту связанности» u как компоненту связанности вершины a в графе, содержащем только ребра с пропускной способностью (емкостью) $\geqslant x$. Нахождение максимального уровня удовольствия, достижимого из a довольно просто; мы можем просто вывести $enj[u]$. Для подсчета второго значения (так как только одна вершина имеет максимальный уровень удовольствия ($mxi[u]$)), мы можем найти максимальное ребро на пути из a в $mxi[u]$ используя двоичные подъемы. (Обозначим это ребро как $\text{maxEdge}(a, mxi[u])$.)

Теперь рассмотрим первоначальную задачу, с возможно равными уровнями удовольствия. И здесь мы делаем ключевое наблюдение: для каждого запроса, максимальное платное ребро всегда лежит либо на **всех** путях от вершины a к любой вершине с максимальным уровнем удовольствия, либо на пути между какими-то двумя вершинами с максимальным уровнем. Пусть ℓ — это вершина с максимальным уровнем, путь к которой из a содержит максимальное платное ребро. И пусть

m — это любая другая вершина с максимальным уровнем. Путь из ℓ в a полностью содержится в объединении путей из m в a и путем между ℓ и m. А потому, максимальное платное ребро лежит хотя бы на одном из этих ребер.

Используя данное наблюдение, мы можем модифицировать DSU, чтобы обрабатывать общий случай. Для начала, пусть $mxi[u]$ равно номеру любой вершины с максимальным уровнем в u. Мы также добавим еще одну переменную $tol[u]$, которая хранит максимальное платное ребро среди всех путей между вершинами с максимальным уровнем в компоненте u. Теперь, когда сливаем компоненты u и v, если $enj[u]$ не равен $enj[v]$, то мы просто берем все значения из компоненты с большим enj. Однако, если $enj[u] = enj[v]$, нам только нужно обновить значение $tol[u]$. Чтобы это сделать, нам нужно учесть как ребра, которые потенциально могут соединить две компоненты, так и те, что соединяют вершины внутри одной компоненты. А потому мы считаем $tol[u] = max(tol[u], tol[v], \text{maxEdge}(mxi[u], mxi[v]))$. И снова, maxEdge можно посчитать с помощью двоичных подъемов.

Для обработки запросов, пользуемся нашим наблюдениям. Для вершины со стартовой вершиной a в компоненте связанности u, максимальный уровень удовольствия все еще равен $enj[u]$. Однако, второе число теперь можно посчитать как $\max(\text{maxEdge}(a, mxi[u]), tol[u])$.

Так как подготовка, необходимая для двоичных подъемов занимает $O(n \log n)$ времени, и все запросы обрабатываются за $O((n+q) \log n)$ времени, то общая асимптотика решения — $O((n+q) \log n)$, что достаточно быстро.