Разбор задачи «Параллельные вселенные»

Назовём граф, который надо сделать связным первым, а граф, связность которого нельзя нарушать, вторым.

Предположим, что второй граф содержит ребро (v, u) такое, что вершины v и u лежат в одной компоненте связности первого графа. Тогда рассмотрим любую вершину a лежащую в другой компоненте связности первого графа. Заметим, что одна из операций (v, a) и (u, a) не нарушает связность второго графа. Таким образом, можно сделать одну операцию так, чтобы компонента вершины a объединилась с компонентой вершин v и u, что позволяет решить задачу.

Теперь заметим, что если существует пара вершин v и u, что в обоих графах нет ребра (v, u), то можно сделать операцию с этой парой вершин, после чего задача сведётся к предыдущему случаю.

Если такого ребра тоже не нашлось, то все компоненты связности первого графа — клики, а второй граф содержит все рёбра, которых нет в первом графе.

Теперь выделим любую компоненту связности первого графа, размер которой не меньше 2 (если такой нет, то сделаем операцию с любой парой вершин, после чего такая компонента связности появится). Выделим в этой компоненте связности произвольные вершины a и b. Кроме этого из всех других компонент связности первого графа выделим по одной вершине $v_1,\ v_2,\ \ldots,\ v_k$.

Заметим, что если первый граф содержит хотя бы три компоненты связности или каждая компонента связности имеет размер хотя бы 2, то подойдёт такая последовательность операций: $(a, v_1),\ (b, v_2),\ (b, v_3),\ \ldots,\ (b, v_k)$.

Если это не так, то первый граф содержит изолированную вершину c, в то время, как все остальные вершины образуют клюку. Второй граф в этом случае имеет вид звезды. В этом случае, если $n = 3$, то ответа нет, а иначе можно выбрать любую пару вершин a и b отличных от c и сделать две операции (a, b) и (a, c).

Это позволяет решать задачу за $\mathcal{O}(n + m_1 + m_2)$ или $\mathcal{O}\left((n + m_1 + m_2)\log(m_1 + m_2)\right)$ в зависимости от реализации.

Разбор задачи «Три массива»

Подгруппа 1. $n \leqslant 15$

Переберем все возможные операции, среди них выберем подходящий ответ. Время работы $\mathcal{O}(2^n \cdot n)$.

Подгруппы 3, 5. $D_i = 0$

Значения A_i, B_i могут измениться только при операции минимума. Переберем конечное значение $A_n = c$. Это может быть любое из значений $L_i \leqslant a_0$ или само a_0. Значение B_n определяется как минимум из b_0 и значений R_i таких, что $L_i < c$. Получаем решение за $\mathcal{O}(n^2)$ или $\mathcal{O}(n \log n)$ в зависимости от способа подсчета значения B_n.

Полное решение.

Заметим, что конечные значения имеют вид $A_n = L_p + D_{p+1} + D_{p+2} + \ldots + D_n$ и $B_n = R_q + D_{q+1} + D_{q+2} + \ldots + D_n$ для некоторых $0 \leqslant p, q \leqslant n$. Заменим L_i на $L_i + \sum_{j=i+1}^{n} D_j$, R_i на $R_i + \sum_{j=i+1}^{n} D_j$. Свели задачу к случаю $D_i = 0$. Получаем разные по сложности решения от $\mathcal{O}(n^3)$ до $\mathcal{O}(n \log n)$.

Разбор задачи «Бурёнка и Pether»

Подгруппа 1.

$n, q \leqslant 100$. В этой подгруппе маленькие ограничения на n, q, поэтому можно честно построить граф прыжков и в этом графе независимо для каждого запроса найти кратчайший путь при помощи поиска в ширину. $O(n^2 q)$

Устройство нашего графа

Замечание. Поймем, как выглядит наш граф, а именно: покажем, что из вершины u ребра проведены во все вершины t некоторого отрезка $[u, r_u]$, что $a_t > a_u$.

Доказательство. Для этого давайте уберем ограничение $a_i < a_j$ для прыжка из i в j (теперь второй конец прыжка может быть любым. главное, чтобы нашлась интересная нам последовательность промежуточных вершин). Если при таких прыжках мы можем прыгнуть в какой-то подотрезок вершин, то при исходных единственное дополнительное ограничение будет $a_i < a_j$, что нам и нужно. Пусть существует прыжок $u \to v$. Покажем, что тогда есть прыжок $u \to v - 1$. В вершину v мы попадаем через какую-то последовательность промежуточных вершин. Пусть последняя промежуточная вершина — t. Тогда $v - t \leqslant d$, поэтому либо $t = v - 1$, либо последовательность можно использовать, чтобы прийти в вершину $v - 1$.

Поиск отрезков.

Теперь научимся искать r_i для всех вершин. Будем перебирать a_i по убыванию значения и поддерживать подотрезки «активированных» элементов в СНМ. При «активации» нового элемента его, возможно, нужно объединить с компонентами соседей в СНМ. Также будем поддерживать множество концов «длинных» подотрезков длины $\geqslant d$. Чтобы по этой информации найти нашу правую границу, необходимо найти конец «длинного» подотрезка, ближайший справа к $i+d$ (пусть он равен e), после чего положить $r_i := \max(i+1, e-sz_e+1)+d$, где sz_e — размер подотрезка с концом e. Такой пересчет корректен, так как все ещё не активированные элементы меньше a_i, поэтому по ним мы можем прыгать без ограничений, а единственным препятствием на нашем пути будет «длинный» подотрезок элементов $\geqslant a_i$.

Подгруппа 2.

$n \leqslant 1000$. Зафиксируем вторую вершину запроса. Будем перебирать вершины в порядке *убывания* идентификатора. Чтобы обновить расстояние от текущей вершины перебора u до зафиксирован-

ной v, нужно просто найти минимум на некотором отрезке. Это легко сделать при помощи дерева отрезков. $O(n^2 \log n + q)$.

Подгруппа 3.

$a_n = n, v_i = n$. Самое время сделать одно из главных замечаний в задаче.

Утверждение. Пусть (u, v) — запрос, на который мы хотим найти ответ, и $a_v = n$. Тогда из u сейчас нужно прыгнуть в наибольшее число на отрезке $[u, r_u]$.

Доказательство. Пусть максимум на подотрезке равен x, а вместо него мы прыгнули в y. После этого мы сколько-то раз прыгнули дальше. Рассмотрим первый момент, когда мы попали в $q \geqslant x$: $y \to t_1 \to \ldots \to q$. Если $q = x$, то и правда оптимальнее было бы просто прыгнуть в q. Иначе если $pos_y < pos_x$, то в q можно попасть, используя какой-то суффикс пути $y \to t_1 \to \ldots \to q$ (pos_x — позиция вершины со значением x в массиве). Если $pos_y > pos_x$, то существует путь $u \to g_1 \ldots \to g_k \to y$. У этого пути можно взять суффикс, а потом объединить с путём $y \to t_1 \to \ldots \to q$, чтобы получить путь $x \to g_i \to \ldots \to g_j \to y \to t_1 \to \ldots \to q$, то есть из x есть прыжок в q. Значит, в x прыгать не менее выгодно, чем в y.

Этот забавный факт позволяет нам при фиксированном v и $a_v = n$ построить дерево, все пути в котором будут кратчайшими от u до v. Сами прыжки легко найти деревом отрезков. Ответить на запросы можно при помощи двоичных подъемов или dfs. $O((n + q) \log n)$

Подгруппы 4, 5.

Заметим, что все вышеупомянутые аргументы верны и для $a_v \neq n$ с той лишь разницей, что в этом случае вершин с идентификатором, большим a_v для нас не существует — из них в a_v мы точно не попадем. Оформим это в виде утверждение, но на самом деле его мы уже доказали.

Утверждение. Пусть (u, v) — запрос. Тогда из u сейчас нужно прыгнуть в наибольшее число x на отрезке $[u, r_u]$, что $x \leqslant a_v$.

Таким образом, в этой подзадаче достаточно построить одно дерево, двоичные подъемы на нем и легко ответить на все запросы $O((n + q) \log n)$.

Подгруппы 6, 7.

Продолжим развивать нашу идею: мы знаем, что прыжков не много, значит, можно просто перебирать запросы в порядке возрастания v_i и поддерживать дерево отрезков на максимум. Тогда при ответе на запрос нужно просто искать максимум на каких-то подрезка не более ans раз. $O(n \log n + q \cdot ans \cdot \log n)$.

Подгруппа 8.

Эта подгруппа является, пожалуй, первым намеком на существование в задаче решения за $O((n+q)\texttt{polylog}(n, q))$. Всё это время мы пытались прыгать в наибольшее число, чтобы сократить расстояние. Однако в этой подгруппе расстояние не должно быть кратчайшим.

Утверждение. Пусть (u, v) запрос и нам не важно расстояние. Тогда можно прыгнуть из u в наименьшее число большее a_u на отрезке $[u, r_u]$ или в вершину v.

Доказательство. Пусть из u не достижима v за один прыжок и существовал путь $u \to t_1 \to \ldots \to v$. Покажем, что после прыжка из u в минимальное число большее a_u на $[u, r_u]$ (пусть это вершина p), всё еще будет существовать такой путь. Найдем первое $t_i > p$. Покажем, что из p можно попасть в t_i. В t_i из t_{i-1} мы попали через какие-то промежуточные вершины: $t_{i-1} \to g_1 \to \ldots \to g_k \to t_i$. Очевидно, что из u в t_{i-1} есть прямой прыжок, как и в p. Рассмотрим первое $g_i \geqslant p$. Если $a_{g_i} \geqslant a_p$, то мы можем просто прыгнуть туда из p. Иначе найдем первое $g_j > a_p$. В g_j мы можем по попасть через промежуточные вершины $g_i \to \ldots \to g_j$. Далее либо из g_j можно перейти в g_{j+1} напрямую, либо мы опять ищем первое $g_r > g_j$ и переходим в него через промежуточные вершины. Так как после g_k идет t_i такой процесс точно конечный, поэтому однажды мы совершим необходимый нам переход и вернемся на интересный нам путь.

Осталось лишь, перебирая a_i в порядке убывания, построить нужное нам дерево, насчитать на нем двоичные подъемы и ответить на все запросы за $O((n+q)\log n)$.

Подгруппа 9.

Из всех предыдущих подгрупп становится понятно, что значения чисел играют более важную роль, чем их индексы. Рассмотрим a_i в порядке убывания значений. Разобьем их на блоки по $\sqrt{n}$. При переходе от блока i к блоку $i+1$ строим двоичные подъемы на переходах, ведущих в числа по значению не лежащие в префиксе уже рассмотренных i блоков. То есть на переходах, ведущих в a_j, что $a_j \leqslant n - i\cdot\sqrt{n}$. В переборе сейчас фиксировано некоторое число a_i. Хотим ответить на все запросы вида (u, i). Для этого мы должны прыгать в дереве при помощи двоичных подъемов, каждый такой прыжок оптимален, если ведет в число $\leqslant a_i$ по доказанному ранее. Значит, при ответе на запрос мы должны будем не более $\sqrt{n}$ раз воспользоваться двоичными подъемами (мы будем допрыгивать в самый левый элемент, из которого в дереве ребро ведет в число, большее a_i, после чего прыгнем ровно 1 раз при помощи дерева отрезков. Так нужно сделать не более $\sqrt{n}$ раз, так как различных «плохих» элементов, в которые мы можем захотеть прыгнуть в двоичных подъемов не больше, чем число элементов в нашем блоке, то есть $\sqrt{n}$. Итого $O((n+q)\sqrt{n}\log n)$.

Подгруппы 10,11.

Стандартные техники работы с корневой позволяют ускорить предыдущее решение до $q\sqrt{n\log n}$ или $q\sqrt{n}$, если использовать вместо двоичных подъемов прыжки по дереву сразу на $\sqrt{n}$ вверх, а вместо дерева отрезков — корневую декомпозицию.

Подгруппа 12.

В подгруппе 8 мы научились решать задачу о существовании пути $O((n+q)\texttt{polylog}(n,q))$. Самое время за аналогичную асимптотика научиться решать и исходную задачу. Итак, мы хотим избавиться от корневой декомпозиции. Попробуем заменить её на метод «разделяй и властвуй». Напишем функцию `solve(l, r, queries)`, которая будет отвечать на все запросы, у которых $a_v \in [l, r)$. Все запросы, у которых $a_u < a_v < m$ или $m \leqslant a_u < a_v$, обработаем при рекурсивных запусках. Нужно как-то разобраться с запросами, пересекающими разрез. Рассмотрим запрос (u, v), пересекающий разрез. Рассмотрим наименьший индекс j, что $j \geqslant u$ и $a_v \geqslant a_j \geqslant m$. Заметим, что на оптимальном пути из u в v будет какая-то вершина, из которой можно будет совершить прыжок в j. Понятно, что до такой вершины. мы можем допрыгать при помощи двоичных подъемов. Прыгнем из этой вершины при помощи дерева отрезков. После такого прыжка мы попадем в какую-то вершины с идентификатором не менее m, так как из нашей вершины точно можно было прыгнуть в j. Заметим, что ранее попасть в вершину с идентификатором не менее m мы не могли, так как j — самая левая из таких вершин. Значит, мы прошли какой-то префикс нашего кратчайшего пути и попали в u', что $a_{u'} \geqslant m$. Изменив таким образом все запросы, пересекающие разрез, мы сведем задачу к двум независимым на половинках. Несложно реализовать обработку таких запросов за $O(q\log n)$ на каждом слое рекурсии. Итоговая асимптотика работы всего решения составит $O((n+q)\log^2 n)$ из-за $\log n$ слоёв рекурсии от «разделяй и властвуй».

Разбор задачи «Почти наверное»

Подгруппа 1.

Для каждого префикса давайте переберем пару чисел, которые будут различаться. После этого выкинем их из массивов и посчитаем сколько операций потребуется, чтобы сделать массивы равными. Число операций равно сумме элементов первого массива минус сумма элементов второго массива. А уровнять их можно только если для каждого i выполнено $a_i \geqslant b_i$. Такое решение работает за $O(n^4)$.

Подгруппа 2.

Давайте улучшим решение прошлой подгруппы. Мы хотим выкинуть по элементу из обоих массивов так, чтобы разница между ними была как можно больше. Давайте отсортируем оба префикса. Заметим, что если мы можем удалить a_i и b_j, то мы можем удалить a_{i-1} и b_j, а также a_i и b_{j+1}. Тогда нам необязательно перебирать $O(n^2)$ пар для удаления, а можно перебирать элемент первого массива, а элемент второго находить двигая указатель. Такое решение работает за $O(n^3)$.

Подгруппа 3.

Снова улучшим решение прошлой подгруппы. Нужно быстрее проверять условие $a_i \geqslant b_i$. Для этого нужно, заметить что b_i элемент сравнивается либо с a_i, либо с a_{i+1}. При чем оба префикса будут разбиваться на $O(1)$ отрезков, в которых нужно делать сравнения одного из двух типов. Давайте префиксными суммами предподсчитаем выполняются ли сравнения каждого из типов, после чего можно будет проводить проверку корректности за $O(1)$. Такое решение работает за $O(n^2 \log n)$.

Идеи для полного решения.

Давайте будем смотреть на два массива, как на набор отрезков $[b_i, a_i]$. Заметим, что нам никогда не выгодно делать в итоговом ответе $a_i < b_i$. Тогда посмотрим как будет выглядить итоговый ответ. Удалим все индексы для которых $a_i = b_i$. Отсортируем оставшиеся числа по возрастанию a_i, тогда $a_1 \leqslant a_2 \leqslant \ldots \leqslant a_n$ и $b_1 < a_1, b_2 < a_2, \ldots b_n < a_n$. Нетрудно понять, что нужно выкинуть a_n и b_1.

Каким условиям должны удовлетворять отрезки, чтобы после удаления a_n и b_1 все было валидно? $b_2 \leqslant a_1, b_3 \leqslant a_2, \ldots, b_n \leqslant a_{n-1}$. Если говорить об этом как о отрезках это значит, что все они образуют связную компоненту. Тогда ответом является сумма длин отрезков минус длина в координатах самой длинной компоненты.

Подгруппа 4-5.

Используя это можно понять, что ответ в 4 подгруппе равен сумме длин отрезков минус длина максимального отрезка. Для пятой группы нужно поддерживать текущую компоненту, при возможности ее расширять и запоминать самую длинную компоненту до этого.

Полное решение.

В реализации полного решения давайте поддреживать текущие компоненты отрезков. При переходе к новому префиксу нужно уметь сливать эти компоненты, если они пересекаются с новым отрезком. Все это можно легко поддерживать в std::set. Время работы решения $O(n \log n)$