Разбор задачи «Нарисуй ломаные»

Для перестановки p обозначим за $a(p)$ количество острых углов среди углов $\angle A_{p_{i-1}} A_{p_i} A_{p_{i+1}}$.

$task = 1$

Нужно найти перестановку p, такую что $a(p) = n - 2$.

Можно за $O(n!)$ перебрать все перестановки.

Для подзадач со случайными точками можно использовать оптимизационные подходы (обсудим позже).

Полное решение этого случая:

- Давайте набирать перестановку, первая точка будет A_1.
- На каждой итерации будем выбирать точку $A_{p_{i+1}}$ (из еще не взятых), такую что длина отрезка $A_{p_i} A_{p_{i+1}}$ максимальная. То есть самую далекую точку.
- Заметим, что $\angle A_{p_i} A_{p_{i+1}} A_{p_{i+2}}$ острый, потому что $A_{p_i} A_{p_{i+1}} \geqslant A_{p_i} A_{p_{i+2}}$.
- Получим перестановку, такую что $a(p) = n - 2$, поэтому это максимум.

Оптимизационные идеи

В подзадачах со случайными точками хорошо работает оптимизация!

Нужно минимизировать функционал $f(p) = -a(p)$ (если $task = 1$) или $f(p) = |a(p) - k|$ (если $task = 3, 4$).

Можно, например, менять случайные 2 элемента перестановки и пересчитывать функционал за $O(1)$. Далее используя локальную оптимизацию или отжиг получится ответ.

Чтобы покрывать все значения k, можно использовать «почти дискретную непрерывность». При локальном изменении функция меняется на $\leqslant 6$. Поэтому можно минимизировать функцию, а потом наоборот ее максимизировать. Таким образом, если запоминать ответы для всех значений, которые получаются, довольно скоро покроются все нужные k.

$x_i < x_{i+1}, y_i < y_{i+1}$

Основная идея про монотонность:

- Допустим у нас есть три точки $A_1(x_1, y_1)$, $A_2(x_2, y_2)$, $A_3(x_3, y_3)$, такие что либо $x_1 < x_2 < x_3$, либо $x_1 > x_2 > x_3$ и либо $y_1 < y_2 < y_3$, либо $y_1 > y_2 > y_3$.
- Тогда угол $\angle A_1 A_2 A_3$ всегда тупой, а углы $\angle A_2 A_1 A_3$ и $\angle A_2 A_3 A_1$ всегда острые.

Тогда работает такое решение:

- Для максимизации просто рассмотрим перестановку $1, n, 2, n-1, 3, n-2, \ldots$.
- Чтобы получать конкретное k подойдет

$$[1, k+2, 2, k+1, 3, k, \ldots], [k+3, k+4, \ldots, n]$$

или

$$[k+2, 1, k+1, 2, k, 3, \ldots], [k+3, k+4, \ldots, n]$$

Монотонные подпоследовательности

- Любую перестановку длины $n \leqslant 2x^2$ можно разделить на $\leqslant 2x$ возрастающих или убывающих подпоследовательностей. Алгоритм:
  - Найдем НВП длины k за $O(n \log n)$.
  - Если $k \leqslant 2x$, то автоматически получаем разбиение на k НУП (например, из алгоритма с бинпоиском).
  - Если $k > 2x$, то удалим эту НВП и перейдем к перестановке длины $\leqslant 2x^2 - 2x - 1 \leqslant (x-1)^2$.
- Общее время построения $O(n\sqrt{n}\log n)$.

$task = 2$

- Нам нравятся монотонные последовательности — можно делать много тупых углов.
- Найдем разбиение множества точек на $\leqslant \sqrt{2n} \leqslant 400$ монотонных подпоследовательностей.
- Можно соединить подпоследовательности, количество острых углов будет $\leqslant 2\sqrt{2n}$.

$task = 3$

- В монотонных подпоследовательностях мы можем получать любые количества острых углов.
- Хочется склеить ломаные между подпоследовательностями. Проблемы будут на границах (непредсказуемое изменение количества острых углов).
- Если в каждой подпоследовательности зафиксировать первый, второй, последний, предпоследний элемент, тогда при склеивании количество острых углов будет предсказуемым.
- Теперь можно будет получать любые k, но определенной четности (из-за фиксирования).
- Если попробовать несколько случайных вариантов, покроются все четности.

$task = 4$

- Делаем такие же примеры.
- Для всех них вместе можно аккуратно посчитать хеши.
- Ранее описанное решение сразу обеспечивает любое $k \in [5\sqrt{2n}, n - 5\sqrt{2n}]$.
- Можно аккуратно улучшить константу до 2, разобрав случаи склеивания.
- Общая асимптотика $O(n\sqrt{n}\log n + qn)$.

Разбор задачи «Доска улик»

Общее замечание: Развернём задачу. Вместо того, чтобы «собирать» дерево с нуля, будем наоборот, удалять рёбра из дерева. А после развернём полученный нами ответ. Далее решаем задачу именно в формате «удалить дерево». Список стикеров в вершине дерева далее будем называть стеком, на вершине стека всегда будет стикер, который должен быть удалён первым.

- **Подгруппа №1**

  Подгруппа решается полным перебором. Есть $(n-1)!$ возможных порядков удаления рёбер. Каждый из порядков тривиально проверяется за $\mathcal{O}(n)$.

  Асимптотика: $\mathcal{O}(n!)$

- **Подгруппа №2**

  Бамбук.

  Динамика по префиксу. $dp[v][i]$ — булевое значение: можно ли удалить бамбук из вершин от 1 до v, но если стек для вершины v содержит единственный элемент $c_{v,i}$ ($i \in \{0, 1\}$).

  База динамики: $dp[1][0] = dp[1][1] = true$

  Пересчёт динамики: надо перебрать все четыре варианта с какими элементами стеков удалится ребро между v и $v-1$, если это ребро можно удалить пересчитываемся через $dp[v-1][i]$.

  Ответ существует, если $dp[n][0] = true$, иначе — нет.

  Чтобы восстановить ответ надо для каждого ребра запомнить с какими i_1, i_2 оно удалялось. Сначала удалить все рёбра вида $i_1 = 0, i_2 = 0$, потом $i_1 = 1, i_2 = 0$, $i_1 = 0, i_2 = 1$ и $i_1 = 1, i_2 = 1$.

  Асимптотика: $\mathcal{O}(n)$

- **Подгруппа №3**

  Звезда.

  Пройдёмся по листам от 2 до n. Пусть сейчас рассматриваем лист v. В массиве c_1 найдём минимальное число $\geqslant w_v - c_{v,1}$. Несложно понять, что именно с этим числом оптимально будет удалить ребро ведущее к v. После чего удалим найденное число из c_1. В реализации поддерживаем c_1 в `std::multiset`, и используем метод `lower_bound`.

  Нужный порядок удаления рёбер соответствует исходному порядку чисел в c_1.

  Асимптотика: $\mathcal{O}(n \log n)$

- **Подгруппа №4**

  Две звезды, основания соединены ребром.

  Решаем две звезды аналогично группе 4. После в обоих основаниях останется по одному числу. Если они позволяют удалить ребро между основаниями, ответ Yes, иначе No. Подходящий порядок удаления рёбер: все рёбра первого ежа до вершины основания, все рёбра второго ежа до вершины основания, ребро между основаниями, все рёбра первого ежа после вершины основания, все рёбра второго ежа после вершины основания.

  Асимптотика: $\mathcal{O}(n \log n)$

- **Подгруппа №5**

  Значения чисел в вершинах возрастают.

  Если в какой-то момент какое-то ребро можно удалить, то и далее его всегда можно будет удалить. А значит в любой момент времени допустимо удалять любое возможное ребро. Итого решением будет жадный алгоритм: ищем любое ребро которое можно удалить и удаляем из дерева, повторяем $n-1$ раз. Если в какой-то момент нет готовых к удалению рёбер, ответ No.

  Асимптотика: $\mathcal{O}(n^2)$

- **Подгруппа №6**

  Заметим, что для произвольного теста, верен следующий факт: если развернуть все стеки оригинального теста и назвать это *новым* тестом, то если для оригинального теста ответа не существует, то не существует и для нового. А если ответ существует, то ответом на *новый* тест будет развёрнутый ответ для оригинального теста. Из этого наблюдения решение группы 6 напрямую следует из решения группы 5: нужно развернуть все массивы, а также итоговый ответ.

  Асимптотика: $\mathcal{O}(n^2)$

- **Полное решение**

  Полное решение основано на одной ключевой идее: «Для любого способа назначить каждому ребру пару элементов стеков из смежных вершин (без повторений), найдется порядок удаления рёбер такой, что каждое ребро будет удалено вместе с назначенными ему элементами».

  Доказательство ключевой идеи: Рассмотрим любое назначение. Каждая вершина указывает на одно смежное ребро, которому назначен верхний элемент стека, как «наполовину готовое к удалению». Так как вершин n, а рёбер $n-1$, найдётся ребро на которое будут указывать дважды, обе смежные вершины. Именно это ребро и можно будет удалить. После удаления дерево распадается на два других, и в них применяется то же рассуждение.

  После ключевой идеи задача распалась на две части.

  1. Построить любое корректное назначение рёбер элементам стеков
  2. Восстановить порядок удаления рёбер

  Для построения назначения используем жадный алгоритм, похожий на решение группы 3. Выпишем порядок вершин любого обхода дерева. Будем перебирать вершины в перевёрнутом порядке обхода (с листьев). Пусть сейчас перебираемая вершина v. Тогда единственному оставшемуся числу в стеке вершины v назначим минимальное подходящее число ($\geqslant w_v - c_v$) из вершины родителя.

  Для реализации будем в каждой вершине хранить `std::multiset` чисел. Будем делать `lower_bound` и `erase` в нём.

  Восстановление: для каждого ребра поддерживаем насколько оно готово к удалению (0/1/2). Рёбра готовые к удалению храним в стеке. Итеративно достаём любое ребро из стека, удаляем его и увеличиваем «готовность к удалению» у не более 2 других рёбер. Если какое-то из них становится готовым к удалению, добавляем его в стек.

  Финальная асимптотика: $\mathcal{O}(n \log n)$

Разбор задачи «Больше подарков хороших и разных»

Подгруппа 1. $n \leqslant 100, k \leqslant 10$.

Используем динамическое программирование.$dp[i]$ = наименьшее количество подотрезков на префиксе длины i, при этом $dp[1] = 1$.

Пересчёт для состояния i: $dp[i] = \min(dp[i], dp[j]+1)$, если подотрезок $[j+1, i]$ содержит не более t различных чисел.

Если пересчитывать данную динамику с конца, то можно проверять условие за $O(1)$, поддерживая количество различных чисел в подотрезке. Таким образом получается решение за $O((nk)^2)$ или $O((nk)^2 \log(nk))$

Подгруппа 2. $t = 1$.

Разобъём исходный массив на блоки одинаковых подряд идущих чисел. Пусть количество блоков будет — cnt. Тогда, ответом будет являться $cnt \cdot k$, если первый блок не совпадает с последним, иначе $(cnt-1) \cdot k + 1$

Доказательство работы Жадного решения.

Рассмотрим некоторое оптимальное разбиение, пусть оно отличается от жадного. Найдём первую позицию в которой есть различия. В таком случае длина подотрезка в оптимальном оптимальном разбиении меньше, чем в жадном. Сдвинем правую границу данного подотрезка до позиции в нашем жадном решении. Разбиение останется корректным и количество подотрезков не увеличится. Продолжая данный процесс, оптимальное разбиение превратится в жадное, причём количество подотрезков не увеличится. Из этого следует, что жадное разбиение является оптимальным.

Подгруппа 3. $n \leqslant 1000, k \leqslant 1000$.

Построим полностью повторённый массив размера nk. С помощью жадного алгоритма наберём минимальное число подарков. Решение за $O(nk)$ или $O(nk \log(nk))$.

Подгруппа 4. $n \leqslant 1500, k \leqslant 10^6$.

Рассмотрим следующий алгоритм — с позиции i в исходном массиве будем жадно набирать подарки до конца текущей стопки. После чего будем добирать из следующей стопки, пока это не увеличивает количество подотрезков. Для каждой позиции i в исходном массиве запомним позицию следующую за той, на которой мы остановимся, а так же количество подотрезков полученных в процессе и назовём эти величины $go[i]$ и $cnt[i]$.

Насчитаем массивы go и cnt за $O(n^2 \log n)$ втупую. Теперь, чтобы посчитать ответ заведём параметр cur, который будет отвечать за позицию в массиве, до которой мы оптимально разбили массив. Идём циклом от 1 до k, на каждой итерации прибавляем к ответу $cnt[cur]$, а cur меняем на $go[cur]$. Подсчёт ответа получается за $O(k)$, прибавляем предпосчёт и получаем $O(n^2 \log n + k)$

Подгруппа 5. $k \leqslant 10^6$.

Научимся находить массивы go и cnt за $O(n)$. Для этого воспользуемся методом двух указателей, чтобы для позиции i находить максимальную длину подотрезка, который содержит не более, чем t чисел. Назовём эту длину — len. Считая массивы go и cnt с конца исходного массива, пересчёт для позиции i можно производить следующим образом:

- Если $i + len \geqslant n - 1$, то $go[i] = (i + len)\%n + 1$, где % операция взятия числа по модулю, а $cnt[i] = 1$
- Иначе, $go[i] = go[i+len+1]$, где % операция взятия числа по модулю, а $cnt[i] = cnt[i+len+1]+1$

Считая ответ как в подгруппе 4, получаем решение за $O(n + k)$ или $O(n \log n + k)$

Полное решение.

Оптимизируем подсчёт ответа. Заметим, что если $k > n$, то при подсчёте ответа величина cur примет некоторые значения несколько раз. Найдём первое повторение. Дальше найдём количество шагов, за которое произошло это повторение, а так же величину прибавленную к ответу за это время. Разделив оставшееся число стопок на количество шагов, найдём сколько ещё раз произойдет повторение, а оставшиеся в конце стопки обработаем отдельно

Итого решение за $O(n)$ или $O(n \log n)$

Разбор задачи «Большая хурма»

Для начала заметим две полезные вещи:

- Максимизировать сумму взятых элементов, это то же самое, что максимизировать разность между тем, что взял ты, и что взял твой оппонент.
- Пока на столе есть ещё хотя бы один кусочек, оба человека съели одинаковое количество, то есть разность между ними равна нулю.

Для начала решим задачу за $O(n^2 \cdot \max\limits_i(w_i))$, используя динамическое программирование:

Обозначим dp[l][r][dif] = разность между тем, сколько возмет первый и тем, сколько возьмет второй, если будут играть на отрезке элементов [l, r], при чем первый начнет есть через dif секунд, после второго (dif может быть отрицательным)

Тогда база динамики: dp[i][i-1][dif] = 0, и ответ лежит в dp[0][n-1][0]

От знака dif зависит, кто будет первым принимать решение о том, какой кусочек взять. Таким образом пересчеты:

Для dif $\leqslant 0$: dp[l][r][dif] = max из

- $dp[l + 1][r][dif + w[l]] + w[l]$
- $dp[l][r - 1][dif + w[r]] + w[r]$

Для для dif > 0: dp[l][r][dif] = min из

- $dp[l + 1][r][dif - w[l]] - w[l]$
- $dp[l][r - 1][dif - w[r]] - w[r]$

В такой динамике $|\text{dif}| \leqslant \max\limits_i(w_i)$, то есть всего $O(n^2 \cdot \max\limits_i(w_i))$ состояний и по 2 перехода из каждого.

Чтобы решить подгруппы с ограничением $w_{i+1} \leqslant 2 \cdot w_i$ докажем следующий факт: в таких ограничениях, для любого достижимого состояния (l, r, dif) выполнено $|\text{dif}| \leqslant w_{r+1}$, (исключение: если $r = n - 1$, то $|\text{dif}| \leqslant w_{n-2}$)

Чтобы это доказать, заметим, что при любом переходе dif меняется в сторону нуля, то есть его модуль либо уменьшается, либо становится не больше, чем w_i (где w_i - последний взятый кусочек). Таким образом, при переходах [l, r] $\to$ [l+1, r] соотношение сохраняется в любом случае, а при переходе [l, r] $\to$ [l, r-1] сохраняется, так как $w_{r+1} \leqslant 2 \cdot w_r$

Таким образом, для каждого l существует всего $O(\sum\limits_{i=0}^{n-1} w_i) = O(W)$ достижимых пар r, dif, то есть всего $O(n \cdot W)$ состояний, и два перехода из каждого.

Теперь, чтобы решить полную задачу, поменяем переходы в динамике так, чтобы они сохраняли соотношение $|\text{dif}| \leqslant w_{r+1}$

Для этого заметим, что если один человек взял несколько кусочков до передачи хода оппоненту, он всегда мог это сделать, сначала взяв самые маленькие кусочки, а затем самые большие. В связи с этим, оставим переход [l, r] $\to$ [l+1, r], так как они сохраняют инвариант, а вместо переходов [l, r] $\to$ [l, r-1], добавим переходы [l, r, dif] $\to$ [l, r', dif'], то есть возьмем столько наибольших элементов, чтобы ход перешел оппоненту, или игра закончилась.

Такие переходы сохраняют инвариант $|\text{dif}| \leqslant w_{r+1}$, то есть в такой динамике $O(n \cdot W)$ состояний, и всё ещё два перехода из каждого.

Заметим так же, что значения r' зависят только от r и dif, но не от l, то есть их можно насчитать за $O(W \cdot \log(n))$.

Полученное решение работает за $O(n \cdot W)$, чего достаточно, чтобы пройти все тесты.

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 $O(n \log n)$