

Задача А. Красивые узоры

Пусть $N(r)$ — количество целых точек, находящихся внутри круга радиуса r .

Можно перебрать все точки в квадрате размером $(2r + 1)^2$ с координатами в диапазоне $[-r, r] \times [-r, r]$ и проверить, выполняется ли условие:

$$x^2 + y^2 \leq r^2$$

Этот метод работает за время $O(r^2)$.

Более быстрый способ — перебрать все x -координаты в диапазоне $[-r, r]$ и подсчитать количество точек на окружности с данным значением x :

$$2\lfloor\sqrt{r^2 - x^2}\rfloor + 1$$

Этот метод работает за время $O(r)$.

В первой подзадаче можно использовать любой метод (медленный или быстрый), чтобы вычислить $N(r)$ для всех радиусов, а затем попробовать все подмножества.

Для второй подзадачи:

1. Вычисляем $N(r)$ с помощью быстрого метода.
2. Используем динамическое программирование, подобное задаче о рюкзаке, с $8 \cdot n$ состояниями.
3. Можно разбить радиусы на группы по остатку от деления на 8 и проверять только первые 7 радиусов из каждой группы (всего 8^8 вариантов).
4. Возможные остатки — только 1 и 5, так что всего существует 8^2 комбинации.

Чтобы быстро вычислить остаток $N(r) \bmod 8$:

1. Убираем все точки, где $x = 0$, $y = 0$ или $|x| = |y|$.
2. Оставшиеся точки можно разбить на 8 эквивалентных классов:

$$(x, y), (x, -y), (-x, y), (-x, -y), (y, x), (y, -x), (-y, x), (-y, -x)$$

3. Количество удалённых точек: $1 + 4r + 4\lfloor r\sqrt{2}\rfloor$

Таким образом, можно вычислить остаток без полного перебора. Время работы: $O(R + n)$.

Задача В. Каменная игра

Эта задача — вариация стандартной игры Ним, но с одним отличием: кучу, из которой нужно брать камни, выбирает противник.

Однако решение не использует теорию Шпрага-Гранди или побитовые операции. Вместо этого используется стандартный подход к разделению состояний на выигрышные и проигрышные.

Определения

- Состояние игры представлено множеством $(a_1, a_2, \dots, a_n)$, где a_i — количество камней в i -й куче.
- Также в состоянии хранится индекс i выбранной противником кучи.

Правила определения состояний:

1. Состояние, в котором все кучи пусты, — проигрышное.
2. Состояние является выигрышным, если текущий игрок может сделать ход, приводящий к проигрышному состоянию.
3. Непустое состояние является проигрышным, если текущий игрок может сделать только такой ход, который ведёт к выигрышному состоянию.

Решение для первых подзадач

Для первой подзадачи можно просто перебрать все возможные состояния $((m+1)^n \cdot n)$ и вычислить каждое состояние с помощью динамического программирования.

Полное решение

Определим:

- A — количество куч с ровно одним камнем.
- B — количество куч с более чем одним камнем.
- H — количество камней в куче, которую указали текущему игроку.

Если $B = 0$ (все кучи содержат один камень):

- Игрок выигрывает, если A нечётно.
- Игрок проигрывает, если A чётно.

Если $B \geq 1$:

Разделим состояния на три типа:

1. Выигрышные: A - чётное.
2. Проигрышные: A - нечётное и $H = 1$.
3. Остальные: A - нечётное и $H > 1$.

Доказательство:

- Если состояние Type-W:
- Если $H \neq 1$ и $B = 1$, можно забрать все камни, оставив A чётным — выигрышное состояние.
- Если $H = 1$, забираем один камень и выбираем другую кучу — переводим соперника в Проигрышное состояние.
- Если $H \neq 1$ и $B > 1$, оставляем один камень, указываем противнику на эту кучу переводя игру для него в Проигрышное состояние.

Если состояние проигрышное, то у игрока нет выбора кроме как забрать единственный камень и перевести игру в выигрышное для соперника состояние.

Остальные состояния зависят от первого хода противника, но они не важны, так как начальное состояние всегда выигрышное по условию.

Реализовать данное решение можно за $O(nm)$.

Задача С. Сортируем перестановку

В первой подгруппе перестановка выглядит хаотично, поэтому для нее ответ всегда $\sqrt{n}$.

Во второй подгруппе можно написать полный перебор за $n! \cdot n^2$ например при помощи рекурсии.

Для дальнейших подгрупп нужно было заметить следующий факт: пусть $[l_1, r_1], [l_2, r_2], \dots, [l_k, r_k]$ операции в их хронологическом порядке, которые сортируют перестановку и их стоимость минимально возможная. Пусть i операция это отрезок X_i , тогда существует такая последовательность операций $X_1, X_2, \dots, X_k$, что $\forall i, j, X_i \cap X_j = \emptyset$ и она имеет также минимальную возможную стоимость.

Доказательство: Возьмем любую корректную последовательность операций, которая имеет минимально возможную стоимость. Докажем по индукции относительно количества пересекающихся пар отрезков, что такой набор можно привести к нужному нам виду.

База индукции: Пересекающихся отрезков нет, тогда набор операций уже в нужном виде.

Переход индукции: Найдем такую пару пересекающихся операций $X_i \cap X_j \neq \emptyset$, что $|i - j|$ минимально возможное. Заметим, что мы можем передвинуть эти операции так, чтобы выполнялось

$|i - j| = 1$, то есть сделать их последовательно. Это возможно, так как между этими операциями нет пересекающихся (иначе бы нашлась пара с меньшим расстоянием индексов), то есть все операции между данными затрагивали другую часть перестановки. Теперь заменим эти две операции на одну операцию $X_i \cup X_j$, покажем, что это никак не поменяло сортировку перестановки. Пусть внутри $X_i \cup X_j$ была инверсия (a, b) , $a > b$, которая с помощью операций X_i, X_j устранялась, тогда она устранится и с помощью операции $X_i \cup X_j$.

То есть количество пар пересекающихся отрезков уменьшилось хотя бы на один. Тогда по предположению индукции на оставшемся наборе операций можно получить нужный вид. Осталось показать, что данная замена не увеличивает стоимость:

$$\begin{aligned}\sqrt{x + y - 1} &\leq \sqrt{x} + \sqrt{y} \\ x + y - 1 &\leq x + 2\sqrt{xy} + y \\ -1 &\leq 2\sqrt{xy}\end{aligned}$$

То есть если бы отрезки пересекались хотя бы по одной точке, то данная замена не увеличила бы стоимость. Соответственно если бы они пересекались в $p > 0$ точках, то $\sqrt{x + y - p} \leq \sqrt{x + y - 1} \leq \sqrt{x} + \sqrt{y}$. После округлений данные неравенства не изменятся.

Чтобы реализовать данную идею воспользуемся динамическим программированием. Пусть dp_i — минимальная стоимость операций, которая сортирует префикс длины i . Чтобы пересчитать данную динамику переберем последнюю операцию, которая заканчивалась в позиции i , а начиналась в позиции j . Тогда данная операция корректна, если $\max(p_j, p_{j+1}, \dots, p_i) = i$, $\min(p_j, p_{j+1}, \dots, p_i) = j$, то есть после сортировки все элементы встанут на свои места. Причем если изначально выполнялось $p_i = i$, то данная операция стоит 0. Тогда пусть операция j корректна: $dp_i = \min(dp_{j-1} + \sqrt{i - j + 1})$. Данное решение работает за $O(n^2)$.

Данная динамика монотонно неубывает (от противного можем отсортировать больший префикс за меньшую стоимость). Данное решение можно как-либо оптимизировать, например заметить что количество различных значений $\sqrt{i - j + 1}$ всего $O(\sqrt{n})$ и получить решения за $O(n\sqrt{n} \log n)$ с помощью разных структур, либо $O(n\sqrt{n})$ на что рассчитаны 4-5 подгруппы. Мы же сразу рассмотрим решение за $O(n)$.

Насчитаем для каждой позиции два значения: $free_i$ и $left_i$. $free_i$ — максимальная длина отрезка начинающегося в точке i который содержит только неподвижные точки. $left_i$ — самый ближайший слева конец «блока». Блок — тот отрезок который после сортировки становится отрезком неподвижных точек (тоже самое что было в динамике за квадрат). Чтобы насчитать $free_i$ достаточно сделать один линейный проход. Чтобы насчитать $left_i$ нужно разбить перестановку на данные блоки, для этого можно идти слева направо по перестановке и каждый раз, когда набирается новый блок оставляем его и начинаем набирать новый. Например:

$$\begin{aligned}P &= [2, 1, 3, 4, 5, 8, 6, 7, 9, 10] \\ left &= [0, 0, 2, 3, 4, 5, 5, 8, 9] \\ free &= [0, 0, 3, 2, 1, 0, 0, 2, 1]\end{aligned}$$

Переделаем динамику, пусть pos_{cost} — максимальный префикс перестановки, который можно отсортировать за стоимость $cost$. Понятно, что в ней всего $\sqrt{n}$ состояний, так как мы всегда можем отсортировать всю перестановку за $\sqrt{n}$ действий.

Будем идти по $cost = 0 \dots \sqrt{n}$, пока не получим $pos_{cost} = n$, тогда $cost$ является ответом на задачу.

Пусть мы находимся на итерации $cost$, тогда сначала мы можем сделать несколько бесплатных операций, за это как раз отвечает массив $free$, то есть сделаем $pos_{cost} += free_{pos_{cost}}$. Далее мы должны совершить какую-то операцию, давайте переберем стоимость данной операции x , теперь возьмем самый большой отрезок с такой стоимостью: $(x + 1)^2 - 1$ и посмотрим, сколько целых блоков мы захватили, за это как раз отвечает массив $left$, он показывает какой самый последний блок мы можем захватить, понятно что взяв в операцию несколько блоков сортировка не сломается, тогда присвоим $pos_{cost+x} = \max(pos_{cost+x}, left_{pos_{cost}+(x+1)^2-1})$. Таким образом мы за стоимость x взяли самый длинный префикс из возможных.

Тогда данное решение работает за $O((\sqrt{n})^2) = O(n)$, что набирает 100 баллов.

Задача D. Вова и самокаты

Для каждого ребра u, v (в обе стороны) хотим посчитать dp_{uv} – количество раз которое Вова, начиная путь в поддереве вершины u относительно корня v , и направляющийся в вершину v оставлял самокат в вершине u .

Очевидно, что тогда ответ это сумма $dp_{uv} \cdot \text{subtree_size}(v, u)$ по всем u, v , где $\text{subtree_size}(v, u)$ – размер поддерева вершины v , относительно корня u .

Чтобы вычислить значения dp_{uv} , построим центроидную декомпозицию и для каждого центроидного поддерева учтём все пути лежащие в нём и проходящие через центроид отдельно.

Рассмотрим ребро x, y веса w . Пусть у вершины x собрались n поездов с остатками заряда в самокатах $r_1 \leq r_2 \leq \dots \leq r_p < w \leq r_{p+1} \dots \leq r_n$, собирающиеся проехать по ребру x, y . (Имеется в виду, что все поездки рассматриваем параллельно, независимо от того что они происходили в разные дни) Тогда первые p самокатов останутся в вершине x и окажутся у вершины y с $K - w$ минутами заряда. Остальные $n - p$ самокатов смогут ехать дальше минут заряда в них уменьшится на w литров.

То есть если рассмотреть оставшиеся заряды как точки на прямой, то точки до w (не включительно) объединятся в точку $K - w$, а все остальные сдвинутся влево на w . Будем хранить заряды упорядоченными по возрастанию в очереди, причём мы будем хранить пары из (заряд, количество самокатов с таким зарядом). Тогда при переходе по ребру нужно будет удалить сколько-то элементов из начала и добавить один в конец.

Для каждого ребёнка центроида хотим вычислить множество зарядов самокатов приехавших из поддерева ребёнка в центроид. Запустим dfs в ребёнка, будем хранить стек пройденных рёбер и префиксные суммы на нём. Для вершины с помощью бинарного поиска по этому стеку найдём номер следующей вершины, в который зарядится самокат выезжавший из этой. Тогда к центроиду он приедет с таким же зарядом, как и самокат выехавший из следующей.

Эта часть работает за $O(n \log n)$ для центроидного поддерева.

Сложим все заряды самокатов в одну очередь и запустим dfs от центроида в поддерева, чтобы учесть вклад в значения dp путей через центроид. При заходе в вершину dfs добавит ребро в очередь, запустится от детей и откатит добавление. Помимо путей через центроид, мы посчитали пути из поддерева до центроида и обратно, их нужно вычесть таким же dfs'ом.

Чтобы уметь откатывать добавление, будем представлять очередь как пару из `std::vector` и указателя на начало. Чтобы добавить ребро, найдём количество удалённых элементов бинарным поиском и вычислим количество самокатов с помощью префиксных сумм. Откат добавления будет тривиальным.

Для центроидного поддерева эта часть работает за $O(n \log n)$.

Всё решение будет работать за $O(n \log^2 n)$.