Разбор задачи «Universal Paperclips»

Главным шагом к решению этой задачи было понять, как меняется суммарное число скрепок между раундами. Промоделируем первый раунд, посчитаем следующие величины:

1. Число улучшений: обозначим за U
2. Число нажатий на кнопку: обозначим за C
3. Число скрепок после первого раунда: обозначим за P

Посмотрим, как будет отличаться второй раунд от первого. Все улучшения будут происходить без изменений, каждое из них увеличит скрепки за каждое нажатие и уменьшит ответ на X. Число нажатий, разумеется, тоже не изменится, однако, каждое нажатие во втором раунде будет приносить на U скрепок больше. Таким образом, во втором раунде мы получим на $U \cdot C$ скрепок больше.

Далее несложно написать формулу для быстрого моделирования $k = \lfloor \frac{t}{n} \rfloor$ раундов необходимо просуммировать арифметическую прогрессию: $P + (P + UC) + (P + 2UC) + \ldots$. Оставшиеся $(t \bmod n)$ секунд можно также промоделировать вручную.

Разбор задачи «Hanoi Chips»

Не уменьшая общности будем считать, что $x_1 \leqslant x_2 \leqslant x_3$ и $y_1 \leqslant y_2 \leqslant y_3$. Заметим, что все перемещения фишек обратимы, поэтому если мы сможем перевести и начальное положение x, и конечное y в одно состояние, то мы сможем также получить решение из x в y. Понятно, что скорее всего не получится получить из начальной позиции любую другую, и всевозможные позиции разобьются на классы. Попробуем придумать какие-нибудь «канонические» положения для x и y. Если канонические положения совпали, то мы получили подходящее решение, а если не совпали — докажем, что решения не существует.

Для начала попробуем поставить фишки как можно ближе друг к другу, то есть минимизировать $x_3 - x_1$ (аналогично будет делать это и следующие шаги и для y). Пока все значения различны, мы можем уменьшать разницу (если $x_3 - x_2 > x_2 - x_1$, то будем передвигать x_1, иначе x_3). Заметим, что при перемещении фишки $\texttt{НОД}(x_3 - x_2, x_2 - x_1)$ не изменяется, поэтому как только мы получим две или более фишек в одной точке, то мы уже получили минимальную разницу (так как $x_3 - x_2$ или $x_2 - x_1$ должно быть не меньше НОД).

В случае, если у нас получилось три фишки в одной точке, то они были все в одной точке изначально (так как все операции обратимы и мы, очевидно, никак не можем сдвинуть фишки). Такой случай обработаем отдельно. Если же у нас в одной точке оказало только две точки, то мы все еще можем двигать этот отрезок (под отрезком понимается пара точек, которая содержит все три фишки), а также его левый конец может содержать одну или две точки. Чтобы однозначно определить каноническое положение, скажем, например, что левый конец должен содержать ровно одну фишку и его координата должна быть минимальна, но больше 0. Чтобы передвигать отрезок, будем просто передвигать один из его концов в нужную сторону.

Переведя x и y в каноническое положение по алгоритму выше, мы можем быстро найти решение, если положения совпали. Осталось доказать, что если положения не совпали, то решения не существует. Действительно, если длины отрезков не сходятся, то это значит, что НОД положений не совпадает, и у нас никак не получится его поменять. Если левый конец не совпал, то у нас никак не получится его немного сместить, потому что x_1, x_2 и x_3 имеют одинаковый остаток по модулю НОД и никогда его не меняют. Пусть два канонических положения положений сдвинуты относительно друг друга ровно на НОД. Представим, что НОД $= 1$ (в противном случае можно сдвинуть и поделить координаты на НОД). Любое перемещение фишек не меняет четность суммы координат, а два положения $(0, 1, 1)$ и $(1, 2, 2)$ имеют разную четность суммы, поэтому ответа для этих двух положений также не существует.

Разбор задачи «Sorting Subarrays»

Подзадача 1

В этой подзадаче можно было написать практически любое решение. Например, перебрать все способы выбрать подотрезок. Потом отсортировать соответствующий подотрезок. Все результаты сложить в какую-нибудь структуру для множеств. Например, `std::set` в C++.

Общая идея

Для всех следующих подзадач нам пригодится общая идея.

Для начала, отдельно учтем исходный массив. Его можно получить, например, применив операцию к любому отрезку длины 1.

Теперь нас интересуют только операции, которые изменяют массив. Пусть мы выбрали для операции отрезок $[l, r]$. Если $a_l = \min\limits_{i=l}^{r} a_i$, то результат применения операции к отрезку $[l, r]$ совпадает с результатом применения операции к отрезку $[l + 1, r]$. Аналогично, если $a_r = \max\limits_{i=l}^{r} a_i$, то результат применения операции к отрезку $[l, r]$ совпадает с результатом применения операции к отрезку $[l, r - 1]$.

Таким образом, будем рассматривать только операции с отрезками $[l, r]$, для которых $a_l > \min\limits_{i=l}^{r} a_i$ и $a_r < \max\limits_{i=l}^{r} a_i$. Такое множество отрезков точно покроет все множество результатов, которые могут дать другие отрезки (кроме результатов, которые не отличаются от исходного массива, но их мы уже учли).

Теперь докажем, что операции со всеми рассматриваемыми отрезками дадут разные результаты. Пусть операции над отрезками $[l_1, r_1]$ и $[l_2, r_2]$ дают одинаковые результаты. Рассмотрим два варианта:

1. $l_1 < l_2$ (не умаляя общности, это покроет случай $l_2 < l_1$). В таком случае, l_1 не принадлежит отрезку $[l_2, r_2]$. Поэтому, при выполнении операции с отрезком $[l_2, r_2]$, a_{l_1} не изменится. С другой стороны, при выполнении операции с отрезком $[l_1, r_1]$ элемент a_{l_1} изменится, потому что он больше минимума на своем отрезке.

2. $l_1 = l_2$. Тогда, не умаляя общности, $r_1 < r_2$. В таком случае, элемент r_2 не принадлежит отрезку $[l_1, r_1]$, и справедливы аналогичные рассуждения для правой границы.

В результате, нам нужно посчитать количество отрезков $[l, r]$, у которых значение в левом конце не равно минимуму, а в правом конце — не равно максимуму.

Подзадача 2

Применим общую идею. Зафиксируем левый конец отрезка, правый будем двигать вправо. При этом, мы сможем поддерживать минимум и максимум на текущем отрезке, а значит и проверить, нужно ли учесть новый отрезок. Время работы $O(n^2)$.

Подзадача 3

Нужно посчитать количество отрезков, которые начинаются на 2 и заканчиваются на 1. То есть, нужно для каждой единицы прибавить к ответу количество двоек слева от нее.

Полное решение

Зафиксируем левую границу отрезка l. Найдем ближайший от нее справа меньший элемент. Обозначим его позицию за p_l. Чтобы для отрезка с левой границей равной l выполнялось условие для левой границы, правая граница должна быть $\geqslant p_l$.

Аналогично, зафиксируем правую границу отрезка r. Найдем ближайший от нее слева больший элемент. Обозначим его позицию за q_r. Левая граница должна быть $\leqslant q_r$.

Массивы p и q можно построить либо с использованием структуры данных, которая умеет вычислять минимум/максимум на отрезке (дерево отрезков, разреженные таблицы, и т.д.), либо с помощью прохода со стеком.

Теперь нам нужно посчитать количество пар $l < r$: $l \leqslant q_r$ и $p_l \leqslant r$. Заметим, что неравенство $l < r$ избыточно, так как оно покрывается неравенством $p_l \leqslant r$.

Нарисуем на плоскости точки (r, q_r). Потом переберем l, и нам нужно будем прибавить к ответу количество точек (x, y), для которых выполнено $x \geqslant p_l$ и $y \geqslant l$. Это стандартная задача «посчитать количество точек в прямоугольнике». Чтобы ответить на такие запросы, можно воспользоваться методом заметающей прямой. Отсортируем точки и запросы по возрастанию x координаты, будем рассматривать их в таком порядке. Будем поддерживать одномерную структуру данных, например дерево отрезков. Если встретили точку, прибавим 1 в структуре данных на позиции y координаты точки. Если встретили запрос, ответом на него является сумма на префиксе структуры данных до y координаты запроса.

Разбор задачи «RestORe»

Решим для начала задачу $n = 1$: найдем, сколько отрезков $[L, R]$ дают искомое побитовое «или» F. Найдем максимальный общий префикс L и R в двоичном представлении: обозначим его P. Тогда $L = P0\ldots$, а $R = P1\ldots$. Среди чисел на отрезке от L до R обязательно будут $P01\ldots1$ и $P10\ldots0$, их побитовое «или» равно $P11\ldots1$. Переберем позицию этого бита, который отличается, проверим, что в F все после этого биты равны единице, и прибавим к ответу число способов выбрать такие L и R: оно равно $2^k \cdot 2^k = 4^k$, где k это число бит после отличающегося. В отдельном случае, когда $L = R = F$, нужно просто прибавить к ответу единицу.

Эти рассуждения не просто дают способ посчитать число таких L и R, но и удобно описывают множество всех подходящих пар: это объединение множеств вида $L \in [P00\ldots0, P01\ldots1]$, $R \in [P10\ldots0, P11\ldots1]$, таких множеств не больше $\log_2 f_i + 1 \leqslant 61$ для каждого значения f_i. Заметим, что для разных длин P эти множества пар (L, R) не пересекаются, поэтому мы не посчитаем что-то дважды.

Решение всей задачи использует динамическое программирование. Состоянием ДП будет пара i и один из не более 61 отрезков, в котором находится правая граница i-го отрезка. Для каждого такого состояния будем считать, сколько есть способов построить i массивов так, что правая граница последнего лежит в этом отрезке. Переход от i к $i + 1$ можно реализовать следующим образом: переберем пару отрезков S_1 и S_2, что $R_i \in S_1$ и $L_{i+1} \in S_2$. Поскольку правая граница i-го отрезка и левая граница $i + 1$-го отрезка должны соседями, то число способов перейти от S_1 к S_2 равно длине их пересечения минус один.

Такое решение можно реализовать за $O(nB^2)$, где B обозначает битовую длину чисел во входе. Такое решение можно реализовать достаточно оптимально, чтобы оно получало полный балл. Однако, можно соптимизировать дальше, и обрабатывать переходы от i к $i + 1$ с помощью двух указателей за $O(B)$.

Разбор задачи «Non-adjacent Swaps»

Для решения этой задачи необходимо было сделать несколько важных наблюдений про описанный в условии процесс.

1. Рассмотрим позиции чисел внутри пар 1 и 2, 2 и 3, $\ldots n-1$ и n. Два числа из любой пары всегда будут находиться в одном и том же относительном порядке, поскольку, чтобы поменяться местами, они будут обязаны в какой-то момент оказаться на соседних позициях.

2. Обозначим за pos_i позицию числа i в перестановке. Набор сравнений $pos_1 \,?\, pos_2$, $pos_2 \,?\, pos3$, $\ldots pos_{n-1} \,?\, pos_n$ является инвариантом (назовем его *профилем* перестановки). Более того, любая перестановка, у которой тот же профиль, является достижимой.

3. Назовем *инверсией* пару элементов, которые находятся в разном порядке в стартовой и финальной перестановках. Тогда из стартовой перестановки можно получить конечную за число обменов, равное числу инверсий между этими двумя перестановками.

Докажем утверждение 3. Расстояние не может быть меньше, че число инверсий, поскольку любой обмен устраняет максимум одну инверсию. Покажем, что такая оценка всегда достижима. Возьмем пару различных перестановок с одинаковыми профилями. Поскольку они отличаются, в них должна быть хотя бы инверсия. Если существует инверсия, элементы которой стоят на соседних позициях в финальной перестановке, их можно поменять местами, что уменьшит как число инверсий, так и расстояние на единицу.

Докажем, что существует такая инверсия. Среди всех инверсий рассмотрим пару (a, b) в которой расстояние между числами в финальной перестановке минимально. Пусть $pos_a < pos_b$ в финальной перестановке, но $pos_a > pos_b$ в стартовой. Если это расстояние равно 1, то мы нашли искомую инверсию. Иначе, возьмем любой элемент c между a и b: тогда, либо (a, c), либо (c, b) образуют инверсию. Почему? В противном случае, если (a, c) стоят в правильном порядке, и (c, b) стоят в правильном порядке, то и (a, b) должны быть в правильном порядке.

Таким образом, два числа, которые необходимо посчитать в задаче, это число перестановок с таким же профилем и суммарное число инверсий стартовой со всеми достижимыми перестановками. Первое число можно вычислить с помощью динамического программирования `dp[i][j]` равно числу способов поставить первые i элементов так, что i стоит в индексе j ($0 \leqslant j < i$, в нумерации с нуля). Для того, чтобы добавить $i + 1$, нужно перебрать, в какое место его вставить, и проверить, что он стоит в правильном порядке относительно i.

Считать инверсии немного сложнее, но общая схема ДП аналогична. Вместо того, чтобы считать число инверсий, зафиксируем одну инверсию $a < b$ и найдем, в скольки перестановках она будет присутствовать. Для этого в предыдущую динамику нужно добавить еще одно измерение, которое будет помнить, на какой позиции находится a. Когда мы дойдем до b, оставим только те переходы, в которых (a, b) образуют инверсию. Такое решение работает за $O(n^6)$: n^2 возможных инверсий, n^3 состояний ДП, переход за $O(n)$.

Переходы в такой динамике можно соптимизировать с помощью префиксных сумм, получив решение за $O(n^5)$. Последняя оптимизация состоит в том, что мы не будем перебирать b, только a. Во время подсчета ДП, когда мы доходим до элемента $b > a$, и эти два элемента образуют инверсию, то необходимо прибавить к ответу число способов вставить оставшиеся элементы. Такая динамика совпадает с динамикой для простого подсчета перестановок (не инверсий), поэтому ее можно предподсчитать однажды для всех a. Это дает решение за $O(n^4)$, которое получает 100 баллов.