Разбор задачи «Вася и коллекция изысканных массивов»

Рассмотрим решение задачи для $n = 2$. Ясно, что в этом случае $x + y = a$, $x - y = b$, где x, y — первый и второй элементы массива. Отсюда $x = \frac{a+b}{2}, y = \frac{a-b}{2}$. В случае, если сумма изначально заданных чисел не делится на 2, то ответа не существует.

Для $n = 3$ попробуем разделить значение $a - b$ примерно на три равные части, а затем к одной из них добавим b. После этого необходимо аккуратно распределить остаток от деления $a - b$ на 3 и проверить, что среди получившихся трех чисел первый и второй максимум всё еще имеют разницу ровно в b.

Еще одним из вариантов решения является перебор первого максимума — max_1. Тогда второй максимум $max_2 = max_1 - b$. Значение a должно быть больше $max_1 + max_2$, а также его должно хватить, чтобы заполнить остальные элементы массива, т.к. они не должны быть равны 0.

Таким образом, мы постепенно подошли к решению задачи на полный балл. Для этого достаточно понять, что в первых $n - 2$ элементах массива можно поставить значение 1. Тогда мы переходим к первой подзадаче, когда у нас осталось распределить $a' = a - (n - 2)$ между двумя числами. Однако при $n > 2$ появляется дополнительный случай: если сумма $a' + b$ не кратна числу 2, то мы можем перенести 1 на любой из элементов, заполненных ранее единицами, но стоит помнить, что этот новый элемент, равный двум, может стать новым вторым максимумом.

Разбор задачи «Пробирки»

Заметим, что каждую пробирку можно представить на плоскости точкой с координатами (a_i, b_i), т.к. третью составляющую всегда можно однозначно определить как $c_i = 10^9 - (a_i + b_i)$. Любая смесь веществ из двух пробирок A и B лежит на отрезке AB, соединяющем точки, представляющие эти пробирки на плоскости. Также и любую смесь, соответствующую точке на отрезке AB можно получить пробирками A и B. Любая смесь веществ из трёх пробирок A, B и C лежит в треугольнике ABC, и наоборот. Первую группу тестов можно было пройти, просто перебирая для каждой точки запроса три точки-вершины треугольника, в котором могла бы находиться точка-запрос. На самом деле одну из вершин треугольника достаточно выбрать произвольной, а не перебирать. Тогда можно пройти вторую группу тестов. Почему это верно, можно понять из дальнейших рассуждений.

Теперь попробуем решить задачу эффективнее. Построим выпуклую оболочку H всех точек, представляющих пробирки. Сейчас её можно построить за $O(N^2)$, но для последующих групп тестов потребуется строить её за $O(N \log(N))$. Понятно, что никакая смесь, представленная точкой P снаружи оболочки, не может быть произведена пробирками: существует прямая l, отделяющая P от H, и любая точка, отрезок или треугольник, в котором лежит точка P, должен лежать по другую сторону от прямой l, чем оболочка, или пересекать прямую. Но тогда такие точка, отрезок или треугольник нам неинтересны, ведь они не могли быть составлены из вершин-пробирок.

Теперь покажем, что хотя бы любую смесь, представленную точкой P внутри или на границе оболочки H, мы можем сделать. Пусть вершины H в порядке обхода по часовой стрелке имеют имена $A_1, \ldots, A_k$. Точка P, если она лежит в H, содержится в одном из треугольников $A_1A_2A_3, A_1A_3A_4, \ldots, A_1A_{k-1}A_k, A_1A_kA_2$. Просто переберём все эти треугольники за $O(N)$. Если ни в один треугольник точка не попала, то она лежит вне H. Иначе выведем в ответ вершины треугольника, в который попала точка-запрос. Такое решение проходит третью группу тестов.

Наконец, научимся проходить четвертую группу тестов. Рассмотрим полярные углы точек $A_2, \ldots, A_k$ относительно точки A_1. Бинарным поиском по массиву значений этих углов найдём, между какими двумя лучами A_1A_i и A_1A_{i+1} лежит луч AP. Теперь понятно, что если точка P и лежит в каком-нибудь из интересных треугольников, то таким треугольником является A_1, A_i, A_{i+1}. Таким образом мы сократили время поиска треугольника с $O(N)$ до $O(\log(N))$. Итоговое время работы $O(M \log(N) + N \log(N))$.

Разбор задачи «Сплошные проблемы»

Задачу можно записать в виде уравнения: $\frac{a+c}{b+c} = d$.

Можно заметить, что для d = 2: $a + c = 2b + 2c$ и $c = a - 2b$.

Если $2b \leqslant a$, то c определяется однозначно. Иначе либо ответа не существует (если $a \neq b$) или ответ 0, если $a = b$.

Разбор задачи «Бить или не бить?»

Самый простой способ решения – перебираем позицию для коня, слона и ладьи, для каждой расстановки проверяем бьёт ли какая-то фигура другую или нет. Такой алгоритм работает за $O(n^3 \cdot m^3)$ и набирает 50 баллов.

Далее, можно убрать перебор одного из измерений – перебираем позицию для коня и для слона, а для ладьи перебираем только номер горизонтали клетки. Тогда нужно проверить, чтобы конь и слон не били друг друга, а потом посчитать сколько клеток в горизонтали ладьи являются подходящими – во-первых, нельзя ставить ладью в столбцы, где стоят конь и слон, во-вторых, нельзя занимать клетки, которые находятся под ударом коня или слона (конь бьёт в горизонтали ладьи не более двух клеток и слон тоже). Остается только посчитать сколько среди таких клеток различных. Остальные являются подходящими позициями для ладьи. Время такого алгоритма $O(n^3 \cdot m^2)$.

Опишем алгоритм за время $O(n \cdot m)$. Пусть A_1 – все подходящие расстановки фигур на доске такие, что все три фигуры не находятся в последнем столбце доски $n \times m$, A_2 – такие расстановки, что все три фигуры не находятся в первом столбце; A_3 и A_4 определяются аналогично для последней и первой строки доски. Сколько всего расстановок, которые принадлежат хотя бы одному из множеств A_1, A_2, A_3, A_4? На помощь приходит формула включений и исключений для четырёх множеств:

$$|A_1 \cup A_2 \cup A_3 \cup A_4| = |A_1| + |A_2| + |A_3| + |A_4| - |A_1 \cap A_2| - |A_1 \cap A_3| - |A_1 \cap A_4| - |A_2 \cap A_3| - |A_2 \cap A_4| - |A_3 \cap A_4| + \\ + |A_1 \cap A_2 \cap A_3| + |A_1 \cap A_2 \cap A_4| + |A_1 \cap A_3 \cap A_4| + |A_2 \cap A_3 \cap A_4| - |A_1 \cap A_2 \cap A_3 \cap A_4|$$

Заметим, что пересечения этих множеств как раз соответствуют расстановкам для досок меньшего размера: например, $A_1 \cap A_2$ – это расстановки фигур на доске без первого и последнего столбца. Поэтому в данной задаче можно применить принцип динамического программирования. Пусть $dp[i][j]$ – число требуемых расстановок на доске размера $i \times j$. Выписав аккуратно все пересечения (например $|A_1| = |A_2| = dp[i][j-1]$), получаем следующую формулу:

$$dp[i][j] = 2 \cdot (dp[i][j-1] + dp[i-1][j]) - dp[i][j-2] - dp[i-2][j] - 4 \cdot dp[i-1][j-1] + 2 \cdot dp[i-1][j-2] + \\ + 2 \cdot dp[i-2][j-1] - dp[i-2][j-2] + X$$

Здесь X – число расстановок, которые не попали ни в одно из множеств A_1, A_2, A_3, A_4. Это в точности такие расстановки, в которых в первой и последней строке, в первом и последнем столбце стоят фигуры. Тогда понятно, что обязательно одна из двух фигур стоит в некотором углу и есть два разных случая – либо в противоположном углу стоит фигура и третья может быть где-то внутри доски, либо в противоположном углу нет фигуры и тогда две оставшиеся фигуры стоят в строке и в столбце, которые не содержат угловую клетку первой фигуры. Перебрав фигуры в этих двух случаях, можно посчитать количество расстановок за $O(1)$ и тем самым найти X.

Разбор задачи «Розовая страна»

Чтобы решение работало верно для первой группы тестов, достаточно перебрать $(N-1)!$ порядков городов и выбрать наилучший. Вторая и третья группа тестов уже требуют полиномиального решения.

Рассмотрим взвешенный граф, вершинами которого являются города, рёбрами — билеты, а весами — цены билетов. Рассмотрим любой простой цикл C в этом графе. При любой покраске городов в розовый и голубой цвета цикл C будет либо полностью состоять из городов одного цвета, либо будет иметь хотя бы два ребра, соединяющих города разных цветов. Это верно, потому что если мы “пройдём” по циклу, то, однажды поменяв цвет города, в котором мы находимся, мы должны будем поменять его обратно.

Из выше показанного следует, что для любого ребра на цикле, если оно в данный момент соединяет города разных цветов, существует ещё одно ребро на этом же цикле, соединяющее города

разных цветов. Тогда ребро e_1 с минимальным весом на цикле $C = \{e_1, \ldots, e_k\}$ никак не влияет на ответ! Удалим такое ребро из графа, сохранив связность. Также мы не изменим вес максимального по весу остовного поддерева в графе. Очевидно, что мы его не увеличим. Может быть мы его уменьшили? Рассмотрим максимальное остовное дерево, и предположим, что оно должно содержать ребро e_1. Удалим ребро из остовного дерева, разбив граф на две компоненты связности. Если бы все рёбра из $\{e_2, \ldots, e_k\}$ соединяли вершины одной компоненты, то и e_1 также соединяло бы вершины одной компоненты, что неверно. Тогда найдётся ребро $e_i, i \neq 1$ такое, что, добавив его, мы восстановим связность остновного дерева. Так как вес e_i не меньше веса e_1, то максимальность остовного дерева мы не изменили. Значит ребро e_1 можно удалить, не изменив вес максимального по весу остовного дерева.

После достаточного числа удалений рёбер по выше указанному правилу мы получим дерево. Конечно, это будет максимальное по весу остовное дерево. Теперь осталось найти ответ для него. Предположим, что за каждый год мы будем получать неудобство, равное не максимуму из весов рёбер, соединяющих города разного цвета, а только весу ребра, один конец которого розовый, а другой конец является тем голубым городом, который мы перекрасим в следующем году. Ясно, что суммарное неудобство при таких правилах не могло бы увеличиться, ведь каждый год мы получаем неудобство, не большее, чем по правилам задачи. Заметим, что суммарное неудобство при новых правилах составляет просто сумму весов всех рёбер, ведь каждый год мы выбирали новое ребро.

Докажем, что полученная нижняя оценка на ответ является достижимой. Будем поддерживать систему непересекающихся множеств (СНМ) на вершинах графа. Изначально каждое множество будет состоять из одной вершины. На множествах будем поддерживать порядок, в котором их следует перекрашивать. Отсортируем рёбра по весу от самого тяжёлого до самого лёгкого, и будем брать рёбра в этом порядке. Очередное ребро будет соединять два множества A и B из СНМ. Для объединённого множества порядок вершин определить очень просто — это будут сначала вершины одного из множеств (пусть A), следом за которыми — вершины другого (пусть B). Заметим, что новое рассмотренное ребро будет входить в ответ только в одном месте: в момент, когда мы перекрасили все вершины из A и собираемся красить вершины из B. Остаётся одно: понять, как сделать так, чтобы столица была перекрашена первой. Для этого достаточно при выборе порядка множеств A и B в объединении всегда первым брать множество, в котором лежит столица. Объединить порядки вершин для множеств можно за константное время, если поддерживать эти порядке в связных списках.

Заметим, что первая часть алгоритма, в которой мы ищем максимальное остовное дерево, и вторая часть, где мы ищем порядок городов, могут быть объединены. Для этого достаточно реализовать алгоритм Краскала и поддерживать в СНМ порядки вершин. Время работы алгоритма $O(M \log(N))$. Такое решение проходит все группы тестов. Не зная алгоритма Краскала или алгоритма Прима для разреженных графов, можно было сдать любой полиномиальный алгоритм, используя идеи выше. Например, можно было написать алгоритм Прима за $O(M + N^2)$.

Разбор задачи «Поиск коробок»

Несложно заметить, что последовательность коробок представляется следующим образом: $1, 2, ...y-1, y+l, y+l+1, ...n, 1, 2, ...$ Покажем, что для решения задачи достаточно найти y: т.к. y - позиция первой потерянной коробки, то номер коробки на ней $y + l$. Чтобы восстановить n достаточно двух запросов: спросим номер коробки на большой позиции (чтобы робот прошел по всем коробкам), после чего спросим номер коробки на позиции, с учетом ответа на предыдущий запрос. Для первой подзадачи достаточно просто перебирать коробки начиная со второй, т.к. первая коробка всегда имеет номер 1. Для второй подзадачи нужно перебирать коробки с шагом 2 и при этом нужно проверять четность y. Для третьей подзадачи воспользуемся корневой декомпозицией. Т.к. позиция $y \leqslant 900$ проверим отсутствие y на интервале от 1 до 30, после чего будем делать запросы, увеличивая позицию на 30. Когда мы найдем первое несоответствие между номером позиции и номером коробки, найдем y на интервале $(x - 30, x]$, где x - последний запрос. Для полного решения воспользуемся методом двоичного подъема. Будем спрашивать номера коробок на позициях равным степеням 2. Несложно доказать, что после нахождения первого несоответствия, длина интервала между предпоследним и последним запросами не превышает количества оставшихся ко-

робок. Таким образом, мы получили интервал $(2^{p-1}, 2^p]$, где p - номер запроса. Так как равенство между номером коробки и позиции сохраняется лишь на части интервала, то для поиска y можно применить бинарный поиск.