Разбор задачи «Лестница для участников олимпиады»

Подзадача 1.

Переберем нижний левый угол лестницы, пусть это клетка (i, j) . Заметим, что в столбце j мы хотим максимизировать количество взятых клеток, так как это будет самый высокий столбец лестницы.

Будем идти по столбцам от j направо, поддерживая высоту последнего столбца лестницы. Теперь нам нужно найти максимальное количество клеток, которое можно набрать в текущий столбец, не превышая высоту последнего столбца. Сделаем это наивно за $O(h)$, получим решение за $O(h^2w^2)$.

Подзадача 2.

Ускорим предыдущее решение, ускорив часть с наивным проходом вверх по таблице. Для этого, предподсчитаем двумерный массив $up_{i,j}$, который будет означать количество подряд идущих единиц в таблице, начиная с клетки (i, j) вверх. Это можно сделать динамическим программированием за $O(hw)$.

Теперь, когда мы перебираем столбец k в решении предыдущей подзадачи, мы можем за $O(1)$ вычислить высоту нового столбца, как минимум из $up_{i,k}$ и высоты последнего столбца.

Получаем решение за $O(h^2w)$.

Подзадача 3.

Заметим, что $\min(h, w) \leq \sqrt{hw}$. Сделаем такое преобразование: повернем таблицу на 90 градусов по часовой стрелке и развернем массив строк (строка i поменяется со строкой $n - i$). При таком преобразовании количество строк и столбцов поменялось местами, а любая лестница изначальной таблицы перешла в лестницу новой, и наоборот.

Тогда решим так: если $h > w$, то сделаем вышеописанное преобразование. Теперь верно, что $h \leq w$, значит решение из второй подзадачи работает за $O(h^2w) = O(hw \min(h, w)) = O(hw\sqrt{hw})$, что укладывается в ограничения подзадачи.

Полное решение.

Для полного решения будем считать площадь лестницы с нижним левым углом в (i, j) с помощью динамического программирования, назовем его $dp_{i,j}$. Найдем высоту первого столбца с помощью массива up , как было описано во второй подзадаче. Теперь лестница устроена так: первые сколько-то столбцов будут иметь высоту $up_{i,j}$, после чего высота уменьшится. Заметим, что столбец, где первый раз высота уменьшится, это ближайший справа индекс k , такой что $up_{i,k} < up_{i,j}$.

Тогда верно, что $dp_{i,j} = up_{i,j}(k - j) + dp_{i,k}$. Осталось эффективно найти значения k для каждой клетки таблицы, это стандартная задача нахождения ближайшего меньшего числа справа для массива высот в каждой строке, которая решается за линейное время проходом со стеком.

Получаем решение за $O(hw)$, которое проходит все подзадачи.

Разбор задачи «Пересменка в Сириусе»

Подзадача 1.

Если какой-то номер не закреплен ни за каким сотрудником, то ответ точно «NO». В первой подзадаче это условие является и достаточным, достаточно направлять работников по убыванию l_i , а при равенстве по убыванию r_i . Тогда все номера, которые за кем-то закреплены будут отремонтированы.

Подзадача 2.

Здесь можно перебрать все возможные порядки направления сотрудников в номера и просимулировать процесс. Это можно сделать за $O(k!kn)$.

Подзадача 3.

Здесь можно перебрать все подмножества номеров, которые уже отремонтированы по возрастанию битовой маски, поддерживая, можно ли добиться именно такого отремонтированного подмножества. Если для достижимой маски перебрать, какого сотрудника надо сейчас направить. Если обновлять маску с помощью битового или, то получится решение за $O(2^n k)$.

Подзадачи 4 – 6.

Здесь нужны разные динамики по подотрезкам. Пусть $dp_{l,r} \in \{0, 1\}$, в зависимости от того, можно ли отремонтировать отрезок номеров с l -го по r -й и только его. Для подсчета $dp_{l,r}$ переберем последнего работника, который что-то отремонтировал в этом сценарии. Пусть его номер i . Тогда должно быть $l \leq l_i$ и $r_i \leq r$ и до его ремонта были отремонтированы все номера на отрезках $[l, r']$ и

$[l', r]$, где $l_i - 1 \leq r' < m_i$ и $m_i < l' \leq r_i + 1$ (если $l = l_i$ или $r = r_i$, то левого или правого отрезка соответственно может и не существовать). То есть должно быть $dp_{l,r'} = 1$ и $dp_{l',r} = 1$.

Если для каждого l и r перебрать все возможные i, l' и r' , то получится решение за $\mathcal{O}(n^4k)$.

Заметим, что l' и r' можно перебирать независимо. Тогда получится решение за $\mathcal{O}(n^3k)$.

Вместо перебора l' надо проверить, существует ли l' в соответствующем полуинтервале, такое что $dp_{l',r} = 1$. Для этого можно параллельно с подсчетом динамики насчитывать префиксные суммы динамики по каждой из координат. То есть нас интересует $pl_{l,r} = dp_{l,r} + dp_{l+1,r} + \dots + dp_{r,r}$ и $pr_{l,r} = dp_{l,r} + dp_{l,r-1} + \dots + dp_{l,l}$. Тогда вместо перебора l' и r' можно проверить, что $pr_{l,m_i-1} - pr_{l,l_i-2} > 0$ и $pl_{m_i+1,r} - pl_{r_i+2,r} > 0$. Тогда дополнительно ничего перебирать не надо и решение работает за $\mathcal{O}(n^2k)$.

Подзадача 7.

Здесь за каждым сотрудником закреплен префикс или суффикс номеров. Если есть два сотрудника, за которыми закреплены префиксы номеров и они оба выполнили свои работы, то одного из них можно было не направлять (того, у кого меньше префикс). Аналогично с суффиксами. То есть достаточно вызвать только двух сотрудников: одного с префиксом и одного с суффиксом. Причем два сотрудника, отрезки которых покрывают все номера не смогут оба отремонтировать свой отрезок только если у них обоих m_i лежит в пересечении их отрезков. Назовим таких двух сотрудников *противоречащими*. То есть можно за $\mathcal{O}(k^2)$ перебрать все пары и проверить.

Подзадачи 8, 9.

Условие этих подзадач на самом деле означало, что если отсортировать отрезки по возрастанию l_i , то и l_i и r_i будут строго возрастать, а еще каждый отрезок не будет целиком покрыт двумя соседними. Назовем сотрудника *полезным*, если когда его направили в номер m_i , этот номер был не отремонтирован (и сотрудник отремонтировал весь свой отрезок). Тогда в этих подзадачах каждый сотрудник должен быть полезным. Для этого для каждой пары соседних сотрудников в порядке сортировки по l_i в пересечении их отрезков должно лежать не более одного m_i этих двух сотрудников (иначе они ни в каком порядке не смогут оба быть полезными). Это условие является и достаточным, если есть k сотрудников, то отрезков пересечения соседних не более $k - 1$, то есть для какого-то сотрудника за его m_i больше никто не ответственен и мы точно сможем направить его последним. Можно его убрать, сделать всех остальных полезными рекурсивно и после этого направить его.

То есть в этой подзадаче надо проверить, что отрезки сотрудников покрывают все номера и что соседние сотрудники не противоречат друг другу.

Подзадача 10.

Здесь тоже если отсортировать отрезки по неубыванию l_i , то r_i тоже будет неубывать, но уже не обязательно делать всех сотрудников полезными. Рассмотрим полезных сотрудников в сценарии где все номера отремонтированы. На самом деле тут тоже достаточно, чтобы никакие два соседних сотрудника не противоречили друг другу. Если какой-то отрезок целиком покрыт двумя соседними, то нетрудно видеть, что эти два соседних тоже не противоречат друг другу. То есть из того, что никакие два соседних не противоречат друг другу следует, что можно выбрать их подпоследовательность, в которой тоже соседние не противоречат и при этом выполняется условие предыдущей подзадачи.

Отсюда получается динамическое программирование: dp_i — можно ли выбрать подпоследовательность сотрудников в порядке возрастания l_j , заканчивающуюся i -м, покрывающую префикс номеров такую, чтобы никакие два соседних сотрудника не противоречили друг другу. Пересчеты можно сделать из всех i во все j с большей левой границей. Надо только проверить, что между соответствующими отрезками нет дырки и что они не противоречат друг другу. Получается решение за $\mathcal{O}(k^2)$.

Подзадачи 11 – 13.

Заметим, что если среди полезных сотрудников есть вложенные отрезки, то внутреннего из них можно было не вызывать. То есть чтобы получить общее решение достаточно в динамику из предыдущей подзадачи добавить условие, что нельзя пересчитываться во вложенный отрезок. То есть мы сортируем отрезки по l_i и пересчитываемся из i -го отрезка в j -й если верно следующее:

- $r_j > r_i \geq l_j - 1$

- либо $m_i < l_j$, либо $r_i < m_j$

Получается динамика за $\mathcal{O}(k^2)$. Также были менее эффективные решения с той же идеей, например $\mathcal{O}(nk)$.

Полное решение.

Теперь надо оптимизировать нашу динамику. Для пересчета в j -й отрезок подойдет любой отрезок, идущий раньше него в порядке l_i , удовлетворяющий условиям из предыдущей подзадачи. То есть у него либо должно быть верно либо $l_j - 1 \leq r_i < r_j$ и при этом $m_i < l_j$ либо просто $l_j - 1 \leq r_i < m_j$. Если завести дерево отрезков на минимум, в котором после обработки i -го отрезка мы ставим на позицию r_i минимум из того, что там уже стоит и m_i , то первый вариант условия проверяется как минимум на отрезке, а второй — как проверка, что на отрезке что-то есть (то есть минимум $< \infty$). Теперь подсчет dp_j выглядит как два запроса к дереву отрезков и одно обновление. Итого решение работает за $\mathcal{O}(k \log n + n)$. Также существовало решение, проверяющее условия с помощью `std::set`.

Разбор задачи «Сочи Парк»

Подгруппы с $x_0 = 0$

Если из точки x_0 пойти вправо до точки $x = x_0 + kd$, то для всех целей, чья координата не превосходит x (обозначим множество их индексов за L) оптимальное количество энергии равно $\min(x_i \% d, d - (x_i \% d))^2$. Обозначим сумму таких значений за $left$. Сумма оптимальных значений для всех координат является решением 1 подгруппы.

Так же для всех целей, чья координата больше x (обозначим множество их индексов за R) посчитаем сумму квадратов координат и сумму координат. Обозначим их за $right2$ и $right$ соответственно.

Тогда для фиксированной точки $x = x_0 + kd$ ответ выглядит следующим образом:

$$\begin{aligned} f(k) &= k \cdot t + \sum_{i=1}^n (x - x_i)^2 = k \cdot t + left + \sum_{i \in R} (x - x_i)^2 = k \cdot t + left + \sum_{i \in R} (x^2 - 2x \cdot x_i + x_i^2) = \\ &= k \cdot t + left + x^2 \cdot |R| - 2x \cdot \sum_{i \in R} x_i + \sum_{i \in R} x_i^2 = \\ &= k \cdot t + left + x^2 \cdot |R| - 2x \cdot right + right2 \end{aligned}$$

Если перебрать все возможные разумные расположения участника, то получится решение за $\mathcal{O}\left(mn \cdot \frac{\max X}{d}\right)$, проходящее подгруппу 4.

Если предварительно отсортировать координаты, то значения $left$, $right$ и $right2$ можно считать префиксными и суффиксными суммами. Тогда переборное решение работает за $\mathcal{O}\left(m \cdot \frac{\max X}{d}\right)$ и проходит 6 подгруппу.

Для решения подгрупп 8, 12 и 14 заметим, что функция является выпуклой, воспользуемся тернарным поиском по количеству перемещений. Чтобы определить конкретные значения $left$, $right$ и $right2$ воспользуемся бинарным поиском или `std::lower_bound`. Получим решение за $\mathcal{O}(m \log n \cdot \log \max X)$. Также для решения подгруппы 8 существует альтернативное решение с вложенными тернарными поисками за $\mathcal{O}(m \log n \cdot \log^2 \max X)$.

Подгруппы с произвольным x_0

Очевидно, что оптимальные перемещения устроены следующим образом: посетить несколько (возможно, ноль) точек с мячами правее x_0 , а затем посетить несколько (возможно, ноль) точек с мячами левее x_0 , или наоборот. При этом чтобы вернуться из части в начало, нужно потратить столько же энергии, сколько на продвижение вперед. Поэтому можно воспринимать первую часть пути с тратой $2t$ калорий.

Заменим все координаты на $x_i - x_0$, а x_0 на 0. Пусть $A = \{x_i : x_i < 0\}$, $B = \{x_i : x_i > 0\}$. Далее можно решать задачу независимо для каждой из частей аналогично решениям, описанным выше. Итоговый ответ равен $\min(\text{solve}(A, t) + \text{solve}(B, 2t), \text{solve}(A, 2t) + \text{solve}(B, t))$, где solve — решение для $x_0 = 0$.

Полные решения

Заметим, что существует не более чем $\mathcal{O}(n)$ позиций координат, в которых меняются значения $left$, $right$ и $right2$, поэтому можно найти с помощью тернарного поиска отрезок, на котором достигается минимум, а затем найти минимум на этом отрезке вторым тернарным поиском. Асимптотика $\mathcal{O}(m(\log n + \log \max X))$. Для любого k из фиксированного отрезка все значения не изменяются и можно вместо тернарного поиска найти вершину параболы по формуле и получить решение за $\mathcal{O}(m \log n)$, которое при аккуратной реализации набирает полный балл.

Для более оптимального решения рассмотрим производную функции f

$$\begin{aligned} f'(k) &= (k \cdot t + left + x^2 \cdot |R| - 2x \cdot right + right2)' = (k \cdot t + left + (x_0 + kd)^2 \cdot |R| - 2(x_0 + kd) \cdot right + right2)' = \\ &= t + 2x_0 d \cdot |R| + 2kd^2 \cdot |R| - 2d \cdot right = t + 2kd^2 \cdot |R| - 2d \cdot right \end{aligned}$$

Теперь можно для поиска отрезка, где достигается минимум, воспользоваться бинарным поиском по производной и смотреть знак производной в точке k , а минимум искать формулой. Асимптотика решения также $\mathcal{O}(m \log n)$.

Разбор задачи «Лягушки на дереве»

Задача требует найти максимальное паросочетание в дереве, где вершины можно брать в пару, если они находятся на нечётном расстоянии не превышающем d .

Подзадача 1.

Можно решить задачу перебором с рекурсией. Пусть текущее множество лягушек (вершин) задано как $\{v_1, v_2, \dots, v_k\}$. Выбираем минимальную вершину и делаем два шага: либо исключаем её из множества, либо подбираем к ней подходящую вершину, находящуюся на нечетном расстоянии не более d , и удаляем обе из множества. Для проверки расстояний можно использовать любой алгоритм поиска кратчайшего пути между всеми парами вершин. В зависимости от реализации асимптотика может быть $\mathcal{O}(n!)$ или $\mathcal{O}(2^n \cdot n)$.

Подзадача 2.

В этой подзадаче можно составлять пару из любых двух вершин, расстояние между которыми нечетное. Заметим, что дерево является двудольным графом. Можно разделить вершины на две доли, например, по четности расстояния до корня. Тогда максимальное паросочетание будет иметь размер, равный минимуму размеров долей, а пары можно восстановить, выбирая любые вершины из разных долей. Решение работает за $\mathcal{O}(n)$.

Подзадача 3.

Эта подзадача сводится к стандартной задаче нахождения максимального паросочетания в дереве. Решение можно получить либо с помощью динамического программирования по поддеревьям, либо с использованием жадного алгоритма.

Подзадача 5.

Здесь достаточно явно построить двудольный граф, в котором две вершины соединены ребром, если они находятся на нечетном расстоянии не более d . После этого можно применить алгоритм поиска максимального паросочетания в двудольном графе, например, алгоритм Куна.

Подзадачи 4, 6–8

В этих подзадачах требуется разработать алгоритм с асимптотикой $\mathcal{O}(nd)$. Основная идея — использование жадного алгоритма. При стандартном поиске в глубину мы пытаемся объединить некоторые свободные вершины из поддерева в пару. Если вершины находятся на расстояниях x и y от текущей вершины, то их можно объединить, если $x + y \leq d$ и они различной четности. Однако наивный жадный подход, который объединяет все такие пары, не работает. Чтобы улучшить алгоритм, заметим, что если $x + y < d$ и вершины различной четности, то их можно будет объединить выше, а не на данной вершине. Таким образом, в каждой вершине следует объединять только те

пары, у которых сумма расстояний равна d . Для этого достаточно хранить множества вершин из поддерева на каждой глубине до d от текущей вершины (например, в двусвязном списке). На каждой вершине мы жадно объединяем пары, расстояние между которыми в сумме равно d . Однако в корневой вершине необходимо запустить отдельный алгоритм жадного поиска, который для каждой вершины подбирает пару с максимально возможным расстоянием до корня.

В зависимости от реализации, можно получить решение за $\mathcal{O}(nd^2)$ или $\mathcal{O}(nd)$.

Полное решение.

Чтобы ускорить алгоритм из предыдущей подзадачи, можно использовать структуру данных для поддержки множества расстояний до текущей вершины для каждой доли, при этом хранить только те расстояния, где еще есть свободные вершины. Также необходимо эффективно определять высоту, на которой в множестве расстояний присутствует пара с суммой d . Для каждого хранимого расстояния x можно поддерживать максимальное y такое, что $x + y \leq d$. При этом добавлять в множество событий высоту, на которой пара даёт сумму d . При аккуратном слиянии таких структур и обновлении событий общее решение будет работать за $\mathcal{O}(n \log n)$.