

Задача 1. Беспилотная аэрологистика

Автор: Алексей Михненко
Разработчик: Алексей Михненко

Подзадача 1.

Замечание: всегда выгодно создать всех роботов в самом начале.

В первой задаче это позволяет перебрать количество роботов, которые в итоге будут созданы, и просимулировать процесс. Поскольку никогда не выгодно создавать роботов больше, чем суммарная высота перегородок плюс максимальная высота робота, такой перебор можно реализовать за $\mathcal{O}(n^3h)$ или за $\mathcal{O}(n^2h)$, где h — максимальная высота.

Подзадача 2.

Во второй подзадаче $n = 0$ (то есть нет препятствий). Отсортируем точки по высоте. Тогда по замечанию из первой подзадачи заказы будут доставлены в какой-то префикс окон. Допустим, мы доставим заказы в k окон и k -е по высоте окно находится на высоте h . Тогда итоговая прибыль будет равна $kp - c(h - 1)$. Параметр k можно перебрать и получить решение за $\mathcal{O}(n \log n)$ из-за сортировки массива.

Подзадача 3.

В третьей подзадаче $n = 1$ (то есть есть ровно одно препятствие). Заметим, что когда роботы встречают препятствие, вместо того, чтобы блокировать часть из них, мы можем считать, что все роботы продолжают движение, но высоты всех окон становятся больше на высоту этого препятствие.

Таким образом, можно увеличить высоты всех окон после первого препятствие на его высоту и свести задачу к предыдущей подзадаче. Получаем решение за $\mathcal{O}(n \log n)$.

Подзадача 4.

В четвёртой подзадаче $m = 1$. Заметим, что для каждого окна можно легко вычислить минимальное количество роботов, которое необходимо создать, чтобы заказ был доставлен. Чтобы сделать это эффективно, нужно к высоте окна прибавить суммарную высоту препятствий до него. Поскольку окно только одно, надо либо не создавать новых роботов, либо создать минимальное количество роботов, чтобы доставить заказ в это окно. Это можно реализовать $\mathcal{O}(n)$.

Подзадача 5.

В пятой подзадаче прибыль от доставки заказов сильно больше, чем стоимость создания новых роботов. Нетрудно видеть, что в этой задаче всегда выгодно создать минимальное количество роботов, чтобы все заказы были доставлены, так как $p > c \sum h_i$. Аналогично предыдущей подзадаче, для каждого окна можно найти минимальное количество роботов, которых достаточно, чтобы заказ был доставлен. Вычисляя максимум по этим значениям, мы получим наименьшее число клонирований, необходимое, чтобы выполнить все заказы. После этого достаточно просимулировать процесс, описанный в задаче, с таким изначальным количеством клонов. Данное решение можно реализовать за $\mathcal{O}(n \log n)$.

Подзадача 6.

Полное решение можно получить, объединив предыдущие идеи. Из третьей подзадачи следует, что все перегородки можно удалить, увеличив при этом высоты всех окон за ними. После удаления перегородок задача сводится ко второй подзадаче. Таким образом, задачу можно решить за $\mathcal{O}(n \log n)$.

Задача 2. 2026

Авторы: Николай Будин, Иван Сафонов, Тихон Евтеев
Разработчик: Валерий Родионов

В подзадаче 1 ограничения на m и n маленькие и $q = 1$, поэтому достаточно промоделировать выполнение одной операции за время $O(\min(n, m) \max(n, m)^2)$.

В подзадаче 2 все операции двигают фишки либо влево, либо вправо. Можно заметить, что в этом случае нас интересует только последнее выполненное движение, поэтому снова достаточно промоделировать одну операцию, но в этой и следующих подзадачах это уже требуется делать за время $O(mn)$.

В подзадаче 3 ограничение на сумму mnq по всем тестам маленькое, поэтому можно просто промоделировать все операции.

В подзадаче 4 нет операций,двигающих фишки вверх. Здесь можно заметить, что после первого движения вверх все фишки «прижмутся» к верху доски, поэтому можно удалить все операции типа «U», кроме самой первой, так как они не будут менять расположение фишек. Теперь единственная операция типа «U» разбивает последовательность операций на два блока, состоящих только из операций «L» и «R», которые мы научились быстро обрабатывать в подзадаче 2.

В подзадачах 5 и 6 подзадаче все буквы, написанные на фишках, совпадают, поэтому фишки можно считать неразличимыми. Пусть s_i — первая операция,двигающая фишки в направлении, перпендикулярном s_1 . Заметим, что после применения последовательности операций $s_1 s_2 \dots s_i$ (ее можно быстро промоделировать с помощью решения подзадачи 2) расположение фишек будет образовывать «неровную лестницу» (не путать с лестницей из 7 подзадачи), то есть они будут прижаты к одному углу таблицы. После этого расположение фишек будет однозначно задаваться количеством фишек в каждой строке и стороной, к которой они прижаты, и будет сохранять такую форму после каждой операции. В подзадаче 5 достаточно моделировать операции с расположением фишек таком формате за $O(mq)$. В подзадаче 6 нужно дополнительно заметить, что расположение фишек не меняется с точностью до угла таблицы, к которому прижаты все фишки, то есть существует всего четыре возможных расположения фишек, поэтому обрабатывать одну операцию можно за $O(1)$.

В следующих подзадачах нам понадобится процесс, который мы назовем *упрощением* последовательности операций. Он будет основываться на следующих фактах:

1. Пусть s_i и s_{i+1} двигают фишки в одном или противоположных направлениях. Тогда по наблюдению из подзадачи 2 операцию s_i можно удалить и ответ не изменится.
2. Пусть s_i и s_{i+2} совпадают, а s_{i+1} двигает в одном из двух перпендикулярных s_i направлений. Тогда операцию по наблюдению из подзадачи 3 операцию s_{i+2} можно удалить и ответ не изменится.

Упрощение будет заключаться в последовательном применении этих двух фактов для удаления ненужных операций, пока это возможно. Этот процесс можно реализовать с помощью линейного прохода со стеком за $O(q)$. Легко видеть, что после упрощения последовательность операций будет иметь период длины 4, равный «LURD» или отличающийся от него поворотом или отражением.

В подзадаче 8 упрощение уже сделано, поэтому там его можно пропустить.

Для простоты будем считать, что q делится на 4. Чтобы получить полное решение, достаточно просто честно выполнить остаток, состоящий из не более чем 3 операций. Также будем считать, что период равен «LURD», так как все остальные получаются из него поворотом или отражением доски.

Назовем комбинированной операцией последовательность из четырех операций «LURD». Упрощенная последовательность операций будет состоять из $q/4$ комбинированных операций.

В подзадаче 7 расположение фишек образует лесенку. Выполним первую комбинированную операцию, после нее лесенка прижмется к правому нижнему углу. Далее каждая комбинированная операция будет крутить лесенку по часовой стрелке, каждый раз возвращая ее в в изначальное положение в правом нижнем углу, но возможно переставляя местами фишки, составляющие лесенку. Порисовав этот процесс на бумаге, можно заметить, что каждые три последовательных комбинированных операции возвращают все фишки в изначальное положение. Таким образом, взяв коли-

чество комбинированных операций по модулю 3, можно свести задачу к применению не более чем двух комбинированных операций.

Для полного решения нужно заметить, что каждая комбинированная операция, кроме первой (которую нужно выполнить отдельно, чтобы прижать все фишки к углу) сохраняет множество клеток доски, на которых стоят фишки. Заменим буквы на фишках на различные целые числа от 1 до k , где k — количество фишек. Промоделируем одну комбинированную операцию и найдем для каждой фишки, на какую позицию она переходит после применения одной комбинированной операции. Получим перестановку p , которая применяется к расположению фишек после выполнения каждой комбинированной операции. Тогда найти финальное расположение фишек можно, применив перестановку p к расположению фишек $q/4 - 1$ раз. Для этого можно использовать бинарное возведение в степень. Получаем полное решение за $O(nm \log q)$.

Задача 3. Кейс на рейс

Автор: Елена Андреева
Разработчик: Владимир Новиков

Данная задача может быть разделена на три независимых задачи с разными c . Решения могут быть разделены на те, которые пользуются позицией кладовых и те, для которых позиция кладовой не имеет значения. Также несложно заметить, что при $k \leq m$ решение всегда существует.

Группы 1, 2, 3, 4. $c = 1$

В данной подгруппе существует простое жадное решение. Будем идти слева направо и поддерживать множество бутылок. Бутылки будут принадлежать к одному из типов: пустые, частично заполненные и без фиксированного типа напитка. Когда мы будем переходить к следующему пассажиру мы будем рассматривать несколько случаев. Если у нас есть частично заполненная бутылка с нужным видом напитка, то мы нальем пассажиру напиток из этой бутылки. Если же нужного типа нет, то для какой-то из бутылок без фиксированного типа напитка мы скажем, что она заполнена этим типом напитка. Такое решение будет ездить к кладовой только в самый поздний из моментов времени, когда это необходимо. Формальное доказательство можно получить из решений на следующие подгруппы, но мы его приводить не будем.

Группы 1, 2, 3, 4. $c = 2$

Решение для $c = 2$ аналогично решению подгруппам с $c = 1$. Единственным отличием является то, что мы запускаем аналогичный жадник, но обрабатываем массив в порядке с последнего элемента до первого.

Группы 1, 5, 9. $n \leq 15, k \leq 15$

Переборное решение не отличается для разных c , поэтому будет набирать неплохие баллы.

В этой подгруппе нужно зафиксировать те позиции, на которых мы будем ездить до кладовых. После фиксации позиций для каждой из них можно выбрать ближайшую из кладовых. Остается проверить, что для этого фиксированного набора позиций возможно заполнить бутылки на кладовых таким образом, что все пассажиры будут обслужены. Это можно сделать одним линейным проходом. После поездки на кладовую мы можем все пустые бутылки пометить «свободными». Если же при дальнейшем проходе нам понадобится взять какой-то из напитков, для которого не существует непустой бутылки, мы возьмем «свободную» бутылку и скажем, что она заполнена этим напитком.

Такое решение будет работать за $O(n \cdot 2^n)$ или $O((n + k) \cdot 2^n)$ в зависимости от реализации.

Группа 3, 7, 11. $p = 1$

Для данных подгрупп достаточно заметить, что каждый раз тележке выгодно покрывать отрезок последовательных сидений. Для подгрупп 3, 7 задачу можно решить формулой, либо жадным образом обрабатывать первые или последние m пассажиров в зависимости от того, покрытие какого отрезка будет дешевле.

Более общим решением будет следующий подход: для каждой позиции i мы можем покрыть отрезок пассажиров с номерами от i до $i + m - 1$ включительно за стоимость поездки тележки от позиции i до какой-то из существующих станций. Тогда задача сводится к следующей: дано множество отрезков, нужно выбрать подмножество минимальной стоимости, чтобы любой из пассажиров

принадлежал хотя бы к одному из выбранных отрезков. Так как множество отрезков имеет линейный размер данная задача может быть решена за $O((n + m) \log n)$ с помощью структур данных. Также данную задачу можно решить с помощью алгоритма Дейкстры. Добавим ребра, соответствующие отрезкам, чтобы они вели из вершины с номером l до вершины с номером r со стоимостью равной стоимости взятия отрезка пассажиров с номерами от l до r и обратные ребра из вершины с номером $i + 1$ до вершины с номером i стоимости 0. Кратчайшее расстояние от вершины 1 до вершины $n + 1$ будет ответом на задачу. Множеством взятых отрезков будет множество ребер, соответствующих отрезкам, которые встретились нам на кратчайшем пути.

Таким образом данную подгруппу можно решать за $O(n \log n)$ с помощью Дейкстры.

Группа 2, 6, 10. $n \leq 2000$

В данной подгруппе будем использовать метод аналогичный подгруппе $p = 1$. Заметим, что для любого префикса u нас фиксируются остатки количеств порций каждого напитка по модулю p .

Для любой границы l посчитаем дальнюю позицию r_l , что мы можем обслужить всех пассажиров на данном отрезке при предположении, что на момент времени l все бутылки пустые, кроме бутылок заполненных остатками по модулю p . Такие бутылки у нас будут обязательно присутствовать вне зависимости от стратегии, так как мы обязаны обслужить всех пассажиров. Данное значение легко посчитать для каждой позиции l за $O(n)$ аналогичным проходом тому, который использовался в переборной подгруппе.

Теперь заметим следующий факт, в случае, если мы можем обслужить пассажиров на отрезке l до r с предположением описанным выше, то мы можем обслужить и пассажиров на отрезке от l до $r - 1$ за ту же стоимость. Тогда мы сводим задачу к задаче покрытия массива отрезками минимальной стоимости, которую мы уже научились решать за $O(n \log n)$. Итоговая асимптотика: $O(n^2)$.

Полное решение.

Для полного решения достаточно научиться искать границы r_l за линейное время. Для этого заметим, что в случае, если мы можем обслужить пассажиров на отрезке от l до r с предположением выше, то мы также можем обслужить пассажиров на отрезке от $l + 1$ до r с таким же предположением. Это наталкивает нас на мысль, что границы можно искать двумя указателями. Корректная реализация работает за $O(n \log n)$ и получает полный балл.

Задача 4. Рамазан и капуста

Автор: Иван Сафонов
Разработчик: Иван Сафонов

Краткое описание задачи: задана фигура на клетчатой плоскости, которая является объединением n прямоугольников. Разрежем фигуру по строкам. Найти множество различных отрезков x координат, которое получится, если спроецировать на ось x полученные кусочки. Также для каждого отрезка нужно найти, в скольких строках он встречается и в каком максимальном количестве строк подряд он встречается.

В первой подзадаче $n = 1$. Ответом является один отрезок $[x_1^L, x_1^R]$ с двумя числами $y_1^R - y_1^L + 1$.

Во второй подзадаче $h = 1$. Заметим, что каждый прямоугольник это просто отрезок и мы должны найти множество отрезков, являющихся объединениями данных отрезков. Это можно сделать, например, с помощью `scanline` по x за $O(n \log n)$.

В третьей подзадаче все ограничения очень маленькие. Можно просто в таблице размера $w \times h$ пометить все клетки фигуры и затем найти все отрезки. Любое полиномиальное решение проходило.

В четвертой подзадаче $w, h \leq 5000$. Заметим, что можно в таблице размера $w \times h$ пометить все клетки фигуры за $O(n + wh)$. Для этого можно, например, прибавить на n прямоугольниках единичку (что делается с помощью префиксных сумм). Затем найти все отрезки можно за $O(wh)$.

Далее, чтобы избавиться от очень больших координат, можно сделать сжатие координат за $O(n \log n)$.

В пятой подзадаче $n \leq 3000$. Вместе со сжатием координат можно сделать такое же решение, как в прошлой подзадаче и получить решение за $O(n^2)$.

В шестой подзадаче $n \leq 10000$. В этой подзадаче можно сделать квадратичное решение, но нужно экономить память. Заметим, что вместо вычисления клеток фигуры, можно делать сканлайн по y и поддерживать массив — «сколькими прямоугольниками сейчас накрыта клетка?» Чтобы вычислять ответ, хочется использовать хеш таблицу, ключами в которых будут отрезки x -ов. Однако можно вместо нее сделать один двумерный массив, который будет хранить индексы отрезков. Время $O(n^2)$, память $O(n)$ или $O(n^2)$ с маленькой константой.

Далее идут подзадачи со специальными условиями.

В седьмой подзадаче все отрезки $[x_i^L, x_i^R]$ пересекаются. Заметим, что это означает, что в каждой строке есть не более одного отрезка. Будем делать сканлайн по y и поддерживать \min и \max x координаты текущих прямоугольников. Таким образом, решение работает за $O(n \log n)$.

В восьмой подзадаче $y_i^L = 1$. Это означает, что фигура имеет вид гистограммы, лежащей на земле. Можно с помощью `scanline` найти эту гистограмму и вывести ответ. Решение работает за $O(n \log n)$.

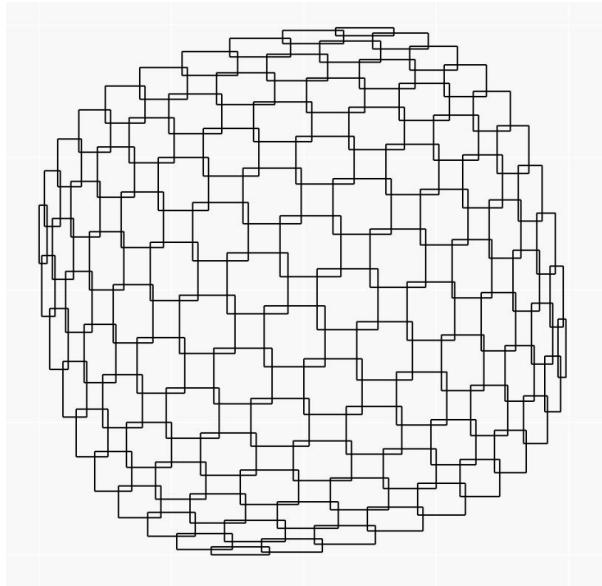
В девятой подзадаче прямоугольники не пересекаются. Давайте будем делать `scanline` по y и поддерживать в `std::set` множество не расширяемых отрезков x , которые находятся в текущей строке. При добавлении или удалении прямоугольника, нужно добавить или удалить кусочек $[x_i^L, x_i^R]$ в строке. Множество отрезков при этом можно несложно обновлять. Таким образом, можно следить за изменением множества не расширяемых отрезков x и считать ответ. Решение работает за $O(n \log n)$.

В десятой подзадаче никакая строка прямоугольника не покрывает другую строку прямоугольника. Как и в прошлой подзадаче будем делать `scanline` по y и поддерживать в `std::set` множество не расширяемых отрезков. В этой подзадаче отличается только способ обновления этого множества при добавлении или удалении отрезка в строке. Это обновление чуть сложнее, но его так же можно сделать с помощью нескольких `lower_bound` в сетах или в ДО.

Все подзадачи с доп. условиями можно сдать с помощью одного на всех похожего `scanline`. Поддерживаем множество не расширяемых отрезков и обновляем его. Чтобы делать обновления без специальных условий (таких как, например, в девятой или десятой подзадачах) можно просто поддерживать в ДО массив: сколькими прямоугольниками сейчас накрыта x координата? В этом ДО происходят $-1/+1$ на отрезке, также нужно находить первый элемент равный 0 и первый элемент > 0 на отрезке. Можно показать, что в подзадачах 7, 8, 9, 10 количество изменений множества в процессе `scanline` будет $O(n)$, поэтому решение работает за $O(n \log n)$.

Почему это решение не является полным? Рассмотрим такой пример «сеточки»: $\frac{n}{2}$ длинных горизонтальных прямоугольников и $\frac{n}{2}$ длинных вертикальных прямоугольников, которые пересекаются в виде сетки. Если мы будем делать `scanline` по y , то множество отрезков будет $\frac{n}{2}$ раз изменяться из множества размера 1 в множество размера $\frac{n}{2}$ и потом обратно, поэтому такой `scanline` будет слишком медленным.

Исходя из этого возникает следующий вопрос. А почему ответ вообще маленький? У жюри есть не доказанная гипотеза, что ответ имеет размер $\leq 3n$. Жюри умеет строить пример, в котором размер ответа $3n - O(\sqrt{n})$.



Пример теста, в котором в пределе ответ стремится к $3n$

Рассмотрим доказательство того, что ответ имеет размер $\leq 2n \log_2 n$. Мы можем посмотреть на задачу под таким углом: на отрезке y -ов $[y_i^L, y_i^R]$ добавляется отрезок x -ов $[x_i^L, x_i^R]$. Для всех y -ов мы изучаем какие существуют не расширяемые отрезки x -ов. Тогда мы можем применить технику DCP offline на массиве y -ов! Тогда нам нужно уметь только добавлять отрезки. При добавлении создается не более одного нового не расширяемого отрезка, но возможно удаляется много. Но тогда заметим, что за всю историю DCP offline будет создано $\leq 2n \log_2 n$ кандидатов на расширяемые отрезки, поэтому ответ имеет размер $\leq 2n \log_2 n$.

Это доказательство можно доделать до решения за $O(n \log^2 n)$, если хранить отрезки в декартовом дереве и обновлять его при добавлении нового отрезка внутри DCP offline. Также нужно уметь прибавлять константу в дереве, для этого нужно использовать ленивые обновления. Однако, поскольку это решение имеет большую константу, оно скорее всего не может пройти на полный балл, а также нельзя поддерживать четвертый параметр для каждого отрезка.

Полное решение

Для полного решения давайте делать scanline по x . Текущую координату назовем тоже x_{cur} . Слева и справа от текущей прямой scanline $x = x_{cur}$ есть какие-то гистограммы. Рассмотрим два массива $h_R[y]$ и $h_L[y]$ — x координаты концов столбиков в строке y справа и слева от прямой scanline. Как они изменяются?

- Для $h_R[y]$ нам нужно делать **max** значением x_i^R на отрезке $[y_i^L, y_i^R]$ в момент времени $x_{cur} = x_i^L$. Также, чтобы было выполнено, что $h_R[y] \geq x_{cur}$ нужно после этого сделать **max** значением x_{cur} на всем ДО.
- Для $h_L[y]$ нам нужно делать **set** значением x_i^L на отрезке $[y_i^L, y_i^R]$ в момент времени $x_{cur} = x_i^L$. Заметим, что так у нас будут корректные значения $h_L[y]$ только для накрытых сейчас y -ов, но нам это подойдет.

Будем хранить массив $h_R[y]$ в ДО. Базовые операции, которые мы будем выполнять это **max** на отрезке и **min** на отрезке. Чтобы уметь выполнять эти операции, нам нужно использовать технику Segment Tree Beats (можно изучить Part 1 в <https://codeforces.com/blog/entry/57319>), которая поддерживает минимум и второй минимум для отрезков ДО.

Как теперь считать ответ? Что значит, что в координате $x = x_{cur}$ в строке y сейчас образуется не расширяемый отрезок?

- Во-первых должен быть прямоугольник i , такой что $y_i^L \leq y \leq y_i^R$ и $x_{cur} = x_i^R$.

- Во-вторых после всех обновлений $h_R[y]$ в этой координате, должно быть выполнено, что $h_R[y] = x_{cur}$, то есть значение на самом деле равно глобальному минимуму во всем массиве.

Рассмотрим все прямоугольники, заканчивающиеся в x_{cur} и рассмотрим объединение отрезков $[y_i^L, y_i^R]$ для них. Это будет множество непересекающихся отрезков. Зафиксируем такой отрезок $[l, r]$. Теперь мы хотим обновить ответ всеми y , такими что $l \leq y \leq r$ и $h_R[y] = x_{cur}$. Второе условие по-другому означает, что $h_R[y]$ равно минимуму на отрезке и минимум равен x_{cur} .

Посмотрим, какая информация нам нужна, чтобы для всех y на отрезке $[l, r]$, в которых сейчас достигается минимум, выделить:

- Множество значений $h_L[y]$ для них.
- Для каждого из них количество y в которых это значение достигается.
- Информацию про подряд идущие y .

Давайте также в нашем ДО хранить такие величины: для всех y , для которых $h_R[y]$ равно минимуму:

- Минимальное значение $h_L[y]$. Максимальное значение $h_L[y]$.
- Дополнительно по всем таким $h_L[y]$ в которых у нас минимум этих величин хранить: их количество и 3 числа для вычисления второго параметра отрезков (это будет длина максимального префикса, суффикса и отрезка подряд идущих y -ов).

Эту информацию можно объединять. Также ее можно пересчитывать при запросах `set` на отрезке массива h_L с помощью ленивых обновлений.

Чтобы теперь выделять отрезки, давайте будем спускаться в ДО на отрезке $[l, r]$, до тех пор, пока не получим отрезок, в котором:

- Минимум значений $h_R[y]$ все еще равен x_{cur} .
- По всем y в которых достигается минимум $h_R[y]$, минимальное значение $h_L[y]$ равно максимальному значению $h_L[y]$ (критерий того, что все элементы равны). Это означает, что на текущем отрезке ДО все отрезки $[h_L[y], h_R[y]]$ одинаковые! Значит мы можем остановить спуск и добавить в ответ информацию об этих отрезках.

В предположениях на линейность размера ответа можно оценить, что данное решение работает за $O(n \log n)$. Детали подсчета максимальных подряд идущих y -ов усложняют написание полного решения, поэтому можно было сделать решение только с количеством и получить 75 баллов.

Чтобы проще осознать полное решение, нужно понять главную идею: в ДО мы можем хранить некоторую информацию по всем позициям, в которых достигается минимум. Эта информация в данном случае достаточно сложная, однако логика пересчета и ее проталкиваний такая же, как если бы это было бы просто, например, количество.

Задача 5. Восстание газонокосилок

Авторы: Азат Сафиуллин, жюри
Разработчик: Николай Будин

В подзадаче 1 ограничение на n маленькое, что позволяет написать перебор направления каждого робота за $O(2^n)$ и проверить за $O(n)$: покроют ли они в таком состоянии весь газон. Получаем решение за $O(2^n \cdot n)$.

В подзадаче 2 все роботы изначально направлены в правую сторону, следовательно, ответом будет 0, если все отрезки между ближайшими роботами будут покрыты. Иначе мы обязаны в каждом

таким непокрытым до конца отрезке поменять направление правого робота и проверить покрытие всего загона. Получаем решение за $O(n)$.

Заметим, что, после того как мы поменяли направления, не должно быть случая, когда два идущих подряд робота направлены в разные стороны. Поскольку отрезок между ними не будет покрыт. Таким образом, направления роботов должны иметь вид: $1, 1, \dots, 1, 1, -1, -1, \dots, -1, -1$.

В подзадаче 3 можно перебрать самого левого робота i , который в итоге будет направлен в левую сторону, и посчитать (количество $d_j = -1$ для $j < i$) + (количество $d_j = 1$ для $i \leq j$). Также надо проверить, что после перенаправления каждый отрезок будет покрыт. Получаем решение за $O(n^2)$.

В подзадаче 4 и 5 для каждого робота заряда хватает для того, чтобы покрыть левый или правый отрезок. Идея такая же, как и в 3 подзадаче, только нужно оптимизировать подсчет поворотов. Это можно посчитать заранее линейными проходами с начала и с конца. Получаем решение за $O(n)$.

Для полного решения достаточно оптимизировать 3 подзадачу. Как говорилось выше, посчитать количество поворотов можно заранее линейными проходами. Это же можно применить для проверки покрытия префикса и суффикса. Получаем решение за $O(n)$.

Задача 6. Интерактивные переходы

Автор: Валерий Родионов
Разработчики: Валерий Родионов, Владимир Новиков

Перепишем задачу на язык графов: корпуса — это вершины графа, а переходы — неориентированные ребра. Если подсветка на корпусе выключена, то покрасим вершину в белый цвет, а если включена, то в черный.

Изначально все вершины и ребра покрашены в белый цвет (подсветка на всех корпусах и переходах выключена), а переключение подсветки на корпусе будет соответствовать перекраске вершины и последующей перекраске всех ребер, у которых концы имеют одинаковые цвета.

В подзадачах 1 и 2 ограничения на n и m маленькие, поэтому их можно сдать перебором различной степени эффективности. В подзадаче 2 предполагалось решение, которое строит граф на 2^{n+m} возможных раскрасках и ищет в нем путь из полностью белой раскраски в требуемую обходом в глубину или ширину за $O(2^{n+m}nm)$.

В подзадаче 3 все $c_i = 1$, то есть нужно покрасить все ребра в черный цвет. Заметим, что если у какого-то ребра оба конца покрашены в белый цвет, то и оно само будет покрашено в белый цвет, поэтому если существует такое ребро, то ответ точно «NO» (это условие является необходимым, поэтому во всех следующих подзадачах будем считать, что оно выполнено Ю иначе сразу выведем «NO»). В противном случае ответ «YES» и получить требуемую покраску можно с помощью следующего алгоритма: сначала покрасим все вершины в черный цвет (после этого все ребра станут черными), после чего перекрасим каждую вершину в требуемый цвет.

В подзадаче 4 граф представляет собой звезду. Снова заметим, что если у ребра цвета концов совпадают, а цвет ребра не совпадает с цветом концов, то ответ «NO». Покрасим каждое ребро по очереди в нужный цвет, перекрасив оба его конца в этот цвет, после чего перекрасим все вершины в цвета из итоговой покраски.

В подзадаче 5 все ребра удовлетворяют условиям $d_{a_i} = c_i$ и $a_i < b_i$, то есть если ориентировать все ребра в направлении увеличения номера вершины, то цвет каждого ребра будет совпадать с цветом левой вершины. Тогда работает следующий алгоритм: будем перебирать вершины в порядке увеличения номера, пусть сейчас мы перебираем вершину v . Тогда покрасим v и все вершины, в которые из v выходят ребра, в цвет d_v . Легко видеть, что после таких действий все исходящие из v ребра будут иметь правильный цвет и никогда его не изменят.

В подзадачах 6 и 7 граф представляет собой один простой путь. Легко видеть, что работает такой алгоритм: в порядке следования по пути покрасим каждое ребро в требуемый цвет, после чего в обратном порядке покрасим каждую вершину в требуемый цвет.

В подзадачах 8 и 9 граф представляет собой дерево. Посмотрим на любой лист, то есть вершину, с которой соединено ровно одно ребро. Если его требуемый цвет совпадает с требуемым цветом соединенного с ним ребра, то по рассуждению из предыдущей подзадачи мы можем покрасить это ребро в нужный цвет и оно навсегда его сохранит, поэтому можно удалить лист и соединенное

с ним ребро. Если требуемый цвет листа не совпадает с требуемым цветом соединенного с ним ребра, то цвет ребра совпадает с цветом другого конца. Тогда сначала мысленно удалим этот лист и покрасим оставшееся дерево, после чего вернем лист, покрасим его в противоположный требуемому цвет (теперь ребро, соединенное с этим листом, станет правильного цвета), после чего покрасим его в требуемый цвет.

В 10 подзадаче граф представляет собой цикл. Если на цикле есть ребро, цвета концов которого совпадают, то это ребро будет иметь правильный цвет независимо от наших действий, поэтому можно его удалить и свести задачу к пути. Значит цвета вершин на пути чередуются. Для вершину u обозначим через t_u номер последней операции, изменившей цвет вершины u . Пусть есть ребро uv , такое, что $d_u \neq d_v$ и $c_{uv} = d_u$. Тогда $t_u < t_v$, там как в противном случае ребро имело бы другой цвет. Тогда если цвета ребер на цикле чередуются, то мы получим, что для любой вершины u на цикле выполняется $t_u < t_u$, то есть требуемой последовательности операций не существует. В противном случае существует вершина, из которой выходит или входит два ребра и по рассуждению, похожему на использованное в подзадачах с деревом, эту вершину можно удалить и снова свести задачу к решению для пути.

Подзадача 11 сделана для неэффективных реализаций полного решения.

В подзадаче 12 дополнительных ограчений нет. Если есть ребро, у которого цвета концов совпадают, то оно либо покрашено в тот же цвет (в этом случае его можно удалить), либо в противоположный цвет (в этом случае можно сразу вывести «NO»). После этого ориентируем все ребра, как в 10 подзадаче. Мы уже доказали, что если в графе появится цикл, то ответ «NO». В противном случае у вершин графа существует топологическая сортировка. Перенумеруем вершины в соответствии с этой сортировкой и получим, что покраска удовлетворяет всем условиям подзадачи 5, которую мы уже умеем решать.

Задача 7. Гонка дронов

Авторы: Азат Сафиуллин, Иван Сафонов
Разработчики: Иван Сафонов, Азат Сафиуллин

В первой подзадаче $n = 2$. Достаточно пройтись двумя указателями по двум массивам, двигая указатель того дрона, который пролетает следующее расстояние быстрее (учитывая индексы дрона). $O(m)$

Во второй подзадаче достаточно запустить гонку для каждого префикса k . Заводим k указателей и двигаем указатель того дрона, который проходит быстрее остальных (учитывая индексы). Получаем решение за $O(n^3 \cdot m)$.

Для решения третьей подзадачи достаточно оптимизировать поиск дрона, который пролетит следующий отрезок быстрее всех. Это можно сделать простым *Set*. $O(n^2 \cdot m \cdot \log(n))$

В четвертой подзадаче расстояния между точками равны. Очевидно, что первым в гонке все расстояния подряд пролетит дрон с минимальной скоростью. Вторым пролетит дрон с минимальной скоростью из оставшихся и так далее. Таким образом, ответ можно вычислить формулой $\frac{(k-1) \cdot k \cdot m}{2}$, где k - текущий префикс.

В пятой подзадаче все скорости равны. Зафиксируем p — индекс левого вхождения максимального расстояния между ближайшими точками. Очевидно, что все отрезки левее p имеют длину меньше. Поэтому в гонке будет момент, когда все дроны соберутся в точке p . Поскольку никакой дрон не может пролететь отрезок максимальной длины быстрее, чем какой-то дрон, который находится левее p . Понятно, что, как только какой-то дрон пролетит максимальное расстояние, он будет лететь до конца непрерывно. А это эквивалентно решению подзадачи 4. То ответ для префикса k будет равен $(p-1) \cdot k \cdot (k-1) + \frac{(k-1) \cdot k \cdot (m-p+1)}{2}$.

Обратим внимание на рекорды слева направо в строго возрастающем порядке. Нетрудно понять, что при преодолении дроном какого-то рекорда, он же продолжит непрерывно лететь до следующего, поскольку до следующего рекорда будут встречаться отрезки не больше текущего рекорда. Таким образом, можно «сжать» количество отрезков, определив за каждым рекордом количество не больших отрезков за ним. Нетрудно доказать, что количество рекордов не больше $\sqrt{s_m}$.

В шестой подзадаче $n \leq 100$. Сожмем до рекордов. Запустим решение на подзадачу 3. Получаем

решение за $O(n^2 \cdot \sqrt{s_m} \cdot \log(n))$.

В седьмой подзадаче не более 2 различных скоростей. Будем поддерживать ответ на префиксе. Когда добавляется новый дрон, достаточно добавить к ответу количество телепортаций, которое совершит этот дрон и количество вынужденных телепортаций других дронов, которые были созданы новым дроном. Это нетрудно посчитать, разобрав случаи с $t_i = 1$ или $t_i = 2$.

Пусть $d_1 = s_1$ и $d_i = s_i - s_{i-1}$ (для каждого $1 < i$)

Будем поддерживать ответ. Будем его обновлять после добавления нового дрона i .

Есть два случая:

1. когда i вынуждает телепортироваться других
2. когда другие вынуждают телепортироваться i

Переберем рекорд u и будем искать подходящее под условия дроны

1. $t_i \cdot d_m \geq t_j \cdot d_u$, т.е. ищем такие j , что i -й еще не финишировал (эквивалентно непрохождению i -го последнего рекорда).
2. $t_i \cdot d_u < t_j \cdot d_m$, т.е. ищем такие j , что еще не финишировали (эквивалентно непрохождению j -го последнего рекорда).

Такие j можно искать корневой декомпозицией с небольшой модификацией. Добавлений не больше $O(n)$, а количество запросов не больше $O(n \cdot \sqrt{s_m})$. Таким образом, на запрос добавления хотим потратить $\sqrt{t_m}$ запросов, а на запрос $O(1)$.

Данная идея с реализацией проходит 11 подзадач.

Для полного решения давайте отсортируем значения $t_{p_1} \leq t_{p_2} \leq \dots \leq t_{p_n}$. Будем перебирать индексы в порядке возрастания значений t , то есть перебирать p_i . Тогда для всех $1 \leq u \leq m$ будем поддерживать указатели $ptr_1[u]$ — максимальное значение j , такое что $t_{p_i} \cdot d_m \geq t_{p_j} \cdot d_u$ и $ptr_2[u]$ — минимальное значение j , такое что $t_{p_i} \cdot d_u < t_{p_j} \cdot d_m$. Поскольку мы перебираем значения в порядке возрастания, мы можем двигать указатели.

Посмотрим как меняется ответ в момент времени p_i . В неравенствах, написанных выше участвуют все индексы p_j по $j \leq ptr_1[u]$ или $j \geq ptr_2[u]$ по всем $1 \leq u \leq m$. Но среди них нужно оставить только $p_j < p_i$. Давайте сделаем массив длины n , где в индексе x будем хранить добавку, которую дает элемент с индексом x . В этой структуре данных нужно $O(n\sqrt{s_m})$ прибавлять к индексу (при движениях указателей) и n раз считать сумму на префиксе. Тогда если применить для этого массива корневую декомпозицию, получится полное решение.

Мы нигде не использовали, что t_i маленькие. Время работы $O(n(\sqrt{s_m} + \sqrt{n}))$, память $O(n)$.

Задача 8. За связь без перебоев

Автор: Никита Лазарев
Разработчик: Алексей Михненко

Первые две подгруппы можно сдать, просимулировав описанный в задаче процесс.

Для решение за квадрат переберём антенну, которая будет заменена на запасную и найдём нестойкость покрытия для каждого случая независимо. Если антенны зафиксированы, можно для каждого i за $O(n)$ найти величину $nxt(i)$, равную максимальному значению r , что существует антенна, покрывающая город i , а также город $r - 1$.

Построим лес (набор деревьев), где предком i -й вершины будет вершина $nxt(i)$. Если $nxt(i) = n$, то вершина i будет корнем какого-то дерева. Пусть $sz(i)$ — размер поддерева вершины i , если каждое дерево подвешено за вершину с максимальным номером. Тогда для фиксированного города $k > 1$ количество пар $s < t$, что при перемещении от города s в город t произойдёт переподключение при перемещении от $k - 1$ в k , равно $(sz(k) - 1) \cdot (n - k)$. Таким образом, ответ на задачу равен:

$$\sum_{k=1}^n (sz(k) - 1) \cdot (n - k)$$

Таким образом, задачу можно решить за $\mathcal{O}(n^2)$.

В четвёртой группе можно заметить, что антенны не выгодно заменять на антенну x , поэтому задача решается за $\mathcal{O}(n)$.

В пятой группе все $a_i = 0$, поэтому можно перебрать позицию, в которую мы поставим антенну, после чего нестойкость покрытия вычисляется по простой формуле, аналогичной решению, приведённому ранее.

Для решения на полный балл посмотрим, как именно изменяется структура леса при добавлении антенны мощности x в городе i . Нетрудно видеть, что какой-то отрезок вершин (по номерам) переподвешивается к вершине $i + x + 1$. Обозначим этот отрезок, как $[l_i, r_i]$. Легко убедиться, что $l_i \leq l_{i+1}$ и $r_i \leq r_{i+1}$, поэтому такие отрезки можно находить двумя указателями.

Будем перебирать i в порядке возрастания и поддерживать текущий отрезок $[l_i, r_i]$, а так же сумму значений $(sz(k) - 1) \cdot (n - k)$ с учётом того, что у вершин с номерами от l_i до r_i ребро в корень удалено. Чтобы поддерживать такую сумму, необходимо при увеличении правой границы на 1 удалять ребро из $r + 1$ в предка, а при увеличении левой границы — добавлять ребро из l_i в исходного предка вершины l_i обратно.

При увеличении правой границы заметим, что все рёбра на пути от r до корня ещё не удалены, поэтому, чтобы посчитать, на сколько изменится нестойкость, необходимо посчитать произведение размера текущего поддерева вершины r и суммы значений $(n - v)$ по всем предкам v вершины r . Такую сумму можно посчитать заранее за $\mathcal{O}(n)$. Чтобы посчитать размер поддерева, переберём всех непосредственных детей вершины r в исходном лесе и, если ребро в ребёнка ещё не удалено, нетрудно видеть, что все рёбра в поддерева ребёнка тоже ещё не удалены, поэтому достаточно просуммировать размеры поддеревьев таких детей.

При увеличении левой границы на 1 достаточно рассмотреть два случая.

- Если предок вершины p в исходном лесе сейчас является корнем, то добавление ребра в этого предка увеличит ответ на $sz(l) \cdot (n - p)$, так как, аналогично увеличению правой границы, легко заметить, что все рёбра в поддерева вершины l точно не удалены
- Если предок вершины l на данный момент не является корнем, то все рёбра от p до корня исходного дерева, в котором содержался p , не удалены, поэтому ответ можно пересчитать, используя сумму, насчитанную для увеличения правой границы.

Кроме всего вышперечисленного, надо поддерживать суммарный размер всех поддеревьев, у которых ребро в предка удалено. Это можно поддерживать аналогично вышперечисленному при увеличении границ отрезка. Из-за добавления новой антенны все такие поддеревья подвешатся к вершине $i + x + 1$. Используя суммарный размер поддеревьев, посчитать реальное значение нестойкости можно аналогично увеличению левой границы.

Таким образом, получили решение за $\mathcal{O}(n)$.