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 последовательноть 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 таблицу. 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 направляниях. 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 возращая 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 возрвращают 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 $k \leqslant m$ 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. $n \leqslant 15, k \leqslant 15$

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. $n \leqslant 2000$

В данной подгруппе будем использовать метод аналогичный подгруппе $p = 1$. Заметим, что для любого префикса у нас фиксированны 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 скольки 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 $w, h \leqslant 5000$. 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 $n \leqslant 3000$. Вместе со сжатием координат можно сделать такое же решение, как в прошлой подзадаче и получить решение за $O(n^2)$.

В шестой подзадаче $n \leqslant 10000$. 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 $O(n \log n)$.

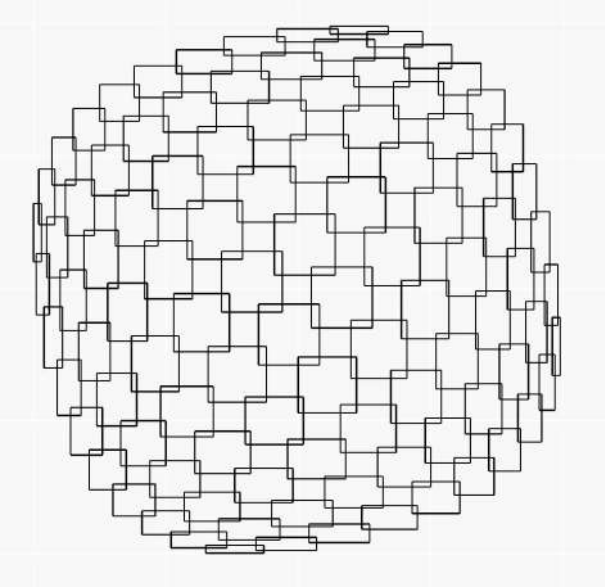
В девятой подзадаче прямоугольники не пересекаются. Давайте будем делать scanline по y и поддерживать в `std::set` множество не расширяемых отрезков x, которые находятся в текущей строке. При добавлении или удалении прямоугольника, нужно добавить или удалить кусочек $[x_i^L, x_i^R]$ в строке. Множество 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 накрывает 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 $\leqslant 3n$. 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 $\leqslant 2n \log_2 n$. 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 отрезок. 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 $\leqslant 2n \log_2 n$ кандидатов на расширяемые отрезки, поэтому ответ имеет размер $\leqslant 2n \log_2 n$.

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 $h_R[y] \geqslant x_{cur}$ 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 $y_i^L \leqslant y \leqslant y_i^R$ 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 $l \leqslant y \leqslant r$ 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.