

## Разбор задачи «Подземелье для принцесс»

Данная задача является задачей на реализацию. Это значит, что нужно написать код, который будет делать ровно то, что просят в условии.

Для того, чтобы после каждой вылазки считать ответ, существует несколько решений. Например, можно разбить задачу по поиску подходящей свободной комнаты на две: поиск свободной комнаты перед входом и после входа. Также при поиске комнаты надо найти ближайшие перед и после входа комнаты двух типов: с вместимостью ровно  $b_j$  и с большей вместимостью.

Затем останется разобрать четыре случая и в зависимости от них вывести ответ. Для удобства после того, как  $i$ -я комната будет выбрана, можно записать  $a_i = 0$ , тогда в дальнейшем данная комната никогда не будет выбрана, так как все  $b_j > 0$ .

## Разбор задачи «Ландшафтный дизайн»

Для начала научимся решать данную задачу, когда  $k = 0$ , то есть требуется за минимальное количество операций сделать все столбики одинаковыми. Суммарное число затраченных операций вычисляется, как сумма разностей между изначальным значением и конечной величиной. Отсюда заметим, что итоговая высота всех столбиков, которую можно достичь за минимальное число операций, это медиана изначальных высот.

Теперь научимся решать для остальных  $k$ . Пусть в итоге получится последовательность  $b_1 = y$ ,  $b_2 = y + k$ ,  $b_3 = y$  и так далее. Тогда заметим, что если из исходных высот на четных позициях, вычесть  $k$ , а потом применить те операции, которые переводят последовательность  $a_i$  в  $b_i$ , получится последовательность из одинаковых чисел. Значит, этот случай решается так же, как при  $k = 0$ .

Случай, когда итоговая последовательность равна  $b_1 = y$ ,  $b_2 = y - k$ ,  $b_3 = y$  и так далее, решается аналогично предыдущему.

Ответ на задачу равен минимуму из ответов для двух данных случаев.

## Разбор задачи «Гонка»

Нетрудно убедиться, что самый выгодный для Марио исход каждого из дополнительных заездов — это исход, в котором персонаж  $v$  занимает первое место, а персонаж  $u$  — последнее. При этом за каждую победу в дополнительном заезде персонаж  $v$  будет получать  $a_1$  баллов в общий зачет, а персонаж  $u$  будет получать 0 баллов, если  $k < n$ , и  $a_k$  баллов, если  $k = n$ .

Также можно заметить, что в случае, когда  $b_i = 0$ , потребуется не более  $m + 1$  дополнительных заездов. Действительно, если сначала в  $m$  дополнительных заездах персонаж  $u$  займет те же места, что занимал в уже прошедших заездах персонаж  $v$ , и наоборот, их общий балл сравняется, так как множества набранных ими баллов будут совпадать. Можно показать, что если в еще одном дополнительном заезде персонаж  $v$  займет первое место, а персонаж  $u$  — последнее, то общий балл персонажа  $v$  станет строго больше общего балла персонажа  $u$ .

### Подзадачи 1 и 2

Для решения первых двух подзадач достаточно было реализовать систему подсчета общего балла, описанную в условии задачи. До тех пор, пока у персонажа  $v$  не больше очков, чем у персонажа  $u$ , будем проводить еще один дополнительный заезд и пересчитывать общие баллы этих двух участников. Асимптотика времени работы этого решения составляет  $O(qm^2s)$  или  $O(qm^2 \log m)$ , в зависимости от реализации процедуры выбора  $s$  минимальных баллов. Баллы за первую подзадачу можно было получить, не реализуя эту процедуру вообще.

### Подзадачи 3 и 4

Для решения вторых двух подзадач можно было улучшить решение предыдущих подзадач, заметив, что при добавлении одного дополнительного заезда можно быстро пересчитывать общие баллы персонажей. Для этого достаточно поддерживать сумму всех баллов персонажа, а также множество из  $s$  минимальных значений баллов, полученных им. При добавлении нового дополнительного заезда это множество меняется одним из двух возможных способов: если балл персонажа за этот заезд не меньше всех значений, лежащих в этом множестве, то оно не изменяется. В противном случае, из этого множества удаляется максимальный элемент, а вместо него добавляется новый балл. Можно также заметить, что для персонажа  $v$ , занимающего всегда первое место, это множество меняться не будет, а для персонажа  $u$ , занимающего всегда последнее место, в множество всегда будет

добавляться новый результат. Реализовав операции добавления и удаления элемента из множества наивно, за время, пропорциональное размеру множества, получаем решение с асимптотикой времени работы  $O(qms)$ . Также для решения данных подзадач необходимо было использовать 64-битный тип данных, чтобы избежать переполнения.

### Подзадачи 5 и 6

Заметим, что если персонаж  $v$  может получить больший общий балл, чем у персонажа  $u$ , за некоторое количество дополнительных заездов, то за любое большее количество дополнительных заездов он тоже сможет этого добиться. Значит, для решения данной задачи можно воспользоваться двоичным поиском по ответу. Для его реализации необходимо быстро проверять для любого числа  $x$ , верно ли, что персонаж  $v$  может получить строго больший общий балл, чем персонаж  $u$ , за  $x$  дополнительных заездов. Однако в данных подзадачах уже не выполнялось ограничение  $b_i = 0$ , а значит, число  $x$  может сильно превосходить число  $m$ .

Изучим внимательнее, как выглядит множество баллов, набранных персонажем, после  $x$  дополнительных раундов. Это множество является объединением множества баллов, набранных им за первые  $m$  заездов, и множества, состоящего из  $x$  одинаковых значений. Обозначим эти одинаковые значения как  $y$  (для персонажа  $v$  выполняется  $y = a_1$ , а для персонажа  $u$  либо  $y = a_k$ , либо  $y = 0$ ). Для того, чтобы получить общий балл, надо из суммы всех элементов этого множества вычесть сумму  $s$  минимальных элементов. Сумму всех элементов этого множества можно легко посчитать: она равна сумме баллов, которые получил персонаж за первые  $m$  заездов (эта величина постоянная и может быть вычислена один раз в начале программы), увеличенной на  $xu$ . Найдем теперь  $s$  минимальных элементов в этом множестве. Для того, чтобы это сделать, достаточно заметить, что в число этих элементов могут попасть лишь  $s$  минимальных элементов из множества баллов, набранных за первые  $m$  заездов, а также число  $y$ , которое будет входить в него не более  $s$  раз. Значит, можно объединить множество, состоящее из  $s$  минимальных баллов за первые  $m$  заездов, со множеством, состоящим из  $s$  элементов, каждый из которых равен  $y$ , и выбрать из получившегося объединения  $s$  минимальных элементов. Это можно сделать за время  $O(s)$ , применив метод двух указателей. Таким образом, можно посчитать общий балл любого персонажа после  $x$  дополнительных раундов и применить эту функцию для проверки в двоичном поиске. Асимптотика времени работы всего решения составляет  $O(qs \log C)$ , где  $C$  — максимальный возможный ответ. В задаче он не превышает  $2 \cdot 10^9$ .

## Разбор задачи «Подарок для Луиджи»

Обозначим за  $L$  максимальную длину палочки.

Для начала заметим, что можно перебрать длины двух сторон прямоугольника и проверить можно ли из данных палочек получить четыре нужные. Сложность такого решения  $O(L^2)$ . Правильно реализованное подобное решение может набирать не менее **24 баллов**.

Далее заметим, что при зафиксированной первой стороне выгодно, чтобы вторая была наибольшей возможной. Более того, если мы можем получить прямоугольник, взяв в качестве второй стороны  $k$ , то мы также сможем получить прямоугольник с взятой второй стороной с длиной менее  $k$ . Также, если из данных палочек невозможно получить четыре палочки для выбранных первой и второй стороны, то также невозможно выбрать большую вторую сторону. Отсюда видим, что для выбора второй стороны можно использовать двоичный поиск. Сложность данного решения  $O(L \log(L))$  и оно набирает не менее **52 баллов**.

Попробуем проанализировать ответ. Существует пять случаев получения сторон прямоугольника из исходных палочек:

- Все четыре стороны получены разломом одной палочки.
- Каждая сторона получена из отдельно взятой палочки.
- Две стороны получены из одной палочки, две из второй.
- Три стороны получены из одной палочки и одна из второй.
- Две стороны получены из одной палочки, остальные две из еще двух.

Рассмотрим все случаи, внутри них рассмотрим варианты взаимного расположения сторон. Выведем наилучший ответ. Сложность данного решения  $O(1)$ . Если рассмотреть все случаи корректно, можно набрать **74 балла**.

Так как  $L$  может быть довольно велико, для оценки площади выбранного прямоугольника нужно использовать умножение и сравнение длинных чисел. Данная модификация предыдущего решения набирает **100 баллов**.

## Разбор задачи «Марио и параллельный мир»

Для того, чтобы решить первую подзадачу достаточно перебрать клетку, в которую встанет Луиджи и воспользоваться методом динамического программирования для подсчета максимального числа очков, которое сможет получить Марио, для каждой выбранной клетки. Сложность такого решения  $O(n^2 \cdot m^2)$ .

Рассмотрим какой-нибудь оптимальный путь, по которому мог бы идти Марио, если бы Луиджи не занимал бы никакую клетку. Заметим, что максимальное количество очков, которое сможет получить Марио, останется неизменным, если Луиджи займет клетку вне данного пути. Сделаем вывод, что стоит перебирать клетки для Луиджи только среди клеток входящих в оптимальный путь. Такое решение работает за  $O((n + m) \cdot n \cdot m)$  и проходит первые две подзадачи.

Наконец заметим, что любой путь проходит ровно через одну клетку каждой побочной диагонали (множество клеток с одинаковой суммой координат). Затем посчитаем максимальные пути от начальной клетки до каждой, а также от каждой до конечной. Тогда для каждой побочной диагонали можно посчитать новый оптимальный ответ, если Луиджи занял клетку в оптимальном пути пересекающую данную диагональ. Для этого среди всех оставшихся клеток на данной диагонали нужно выбрать ту, у которой сумма максимального пути от начальной клетки до нее и от нее до конечной клетки будет максимальной. Чтобы выбирать эту клетку за  $O(1)$  можно для каждой диагонали предподсчитать две наибольшие клетки по данной сумме (наибольшая, очевидно, входит в оптимальный путь). Теперь среди вторых максимумов на каждой диагонали нужно выбрать наименьший, он и будет являться ответом на задачу. Итоговая сложность данного решения  $O(n \cdot m)$ .

## Разбор задачи «Огород Марио»

В задаче требовалось найти клетку внутри прямоугольника, у которой минимальное из манхэттенских расстояний до данного набора клеток максимален. Напомним, что Манхэттенское расстояние между клетками с координатами  $(x_1, y_1)$  и  $(x_2, y_2)$  — это величина  $|x_1 - x_2| + |y_1 - y_2|$ .

### Подзадача 1

В случае, когда изначально заражена только одна клетка, требуется найти внутри прямоугольника клетку с максимальным манхэттенским расстоянием до данной клетки. Можно показать, что такой клеткой всегда будет являться один из углов прямоугольника. Таким образом, для решения данной подзадачи достаточно было найти максимальное манхэттенское расстояние от данной клетки до одного из углов прямоугольника.

### Подзадачи 2 и 3

В этих подзадачах достаточно было промоделировать процесс, описанный в условии задачи. Баллы, набираемые решением, зависят от эффективности реализации моделирования. Также для решения этих подзадач можно было перебрать все клетки внутри прямоугольника и для каждой из них найти минимальное манхэттенское расстояние до одной из данных клеток. Асимптотика времени работы такого решения составляет  $O(nrc)$ .

### Подзадачи 4 и 6

Подзадачи, в которых ограничения на размер прямоугольника позволяют это сделать, можно применить поиск в ширину. Построим граф, вершинами которого будут клетки, а ребром будут соединены все пары клеток, соседних по стороне. В этом графе запустим алгоритм поиска в ширину одновременно из всех изначально зараженных клеток (для этого достаточно все эти клетки изначально поместить в очередь, используемую при обходе). Теперь для каждой клетки мы знаем расстояние до ближайшей из изначально зараженных. Чтобы получить ответ на задачу, выберем из всех этих расстояний максимальное. Асимптотика времени работы такого решения при эффективной реализации поиска в ширину составляет  $O(rc + n)$ .

### Полное решение

Очевидно, что если в какой-то момент все клетки были заражены, то в каждый из последующих моментов они также все заражены. Значит, в задаче можно применить двоичный поиск по ответу. Таким образом, задача сводится к следующей: проверить, верно ли, что через  $t$  секунд после начала процесса все клетки были заражены.

Рассмотрим для каждой изначально зараженной клетки множество всех клеток внутри прямоугольника, манхэттенское расстояние до которых от этой клетки не превосходит  $t$ . Назовем такое множество *окрестностью* клетки. Легко заметить, что множество клеток, зараженных через  $t$  секунд после начала процесса — это объединение всех окрестностей изначально зараженных клеток. Заметим также, что каждая окрестность выглядит как квадрат, повернутый относительно сторон исходного прямоугольника на  $45^\circ$ , у которого расстояние от центра до углов составляет  $t$ .

Чтобы проверить, что объединение окрестностей полностью покрывает прямоугольник, применим преобразование ко всем клеткам: клетку с координатами  $(x, y)$  заменим на точку с координатами  $(x + y, x - y)$ . Так как сумма и разность чисел всегда имеют одну четность, у получившихся точек четность координат будет совпадать. Посмотрим, во что переходит окрестность клетки при таком преобразовании. Можно понять, что она переходит во все точки внутри некоторого квадрата, у которых одинаковая четность координат. Клетки исходного прямоугольника же переходят в точки с одинаковой четностью координат, расположенные внутри некоторого прямоугольника, повернутого относительно осей на  $45^\circ$ . Для простоты мы можем считать, что квадраты содержат в себе не только точки с одинаковой четностью координат, но и все остальные, ведь добавление этих точек никак не поспособствует покрытию необходимого нам прямоугольника, так как всего его точки имеют одинаковую четность координат.

Эта задача похожа на задачу объединения прямоугольников, которая решается с помощью метода сканирующей прямой. Применим схожий метод: запустим сканирующую прямую слева направо и будем следить за событиями, которые происходят. Событием является либо начало (левая сторона) какого-то из квадратов, либо его конец (правая сторона). Для каждой  $y$ -координаты будем хранить, сколькими квадратами покрыта сейчас точка сканирующей прямой с соответствующей координатой. Когда сканирующая прямая встречает начало квадрата, нужно для всех  $y$ -координат, попадающих в этот квадрат, увеличить их счетчики на единицу, а когда прямая встречает конец квадрата — уменьшить. В некоторые моменты времени необходимо проверять, что все точки с текущей  $x$ -координатой, лежащие в прямоугольнике, покрыты хотя бы один раз. Можно доказать, что достаточно осуществлять такую проверку только в положениях сканирующей прямой,  $x$ -координата которых отличается от  $x$ -координаты какого-то из квадратов не более, чем на 1. Таким образом, есть  $O(n)$  положений сканирующей прямой, в которых достаточно осуществить проверку. Каждая проверка заключается в следующем: надо проверить, что внутри какого-то отрезка  $y$ -координат все точки с фиксированной четностью покрыты хотя бы один раз. Границы этого отрезка определяются пересечением интересующего нас прямоугольника и сканирующей прямой и могут быть вычислены за константное время. Четность  $y$ -координат точек внутри отрезка, интересующих нас, определяются четностью текущей  $x$ -координаты: эти четности должны совпадать.

Таким образом, на сканирующей прямой мы хотим делать следующие операции:

- Увеличить все числа на отрезке на единицу.
- Уменьшить все числа на отрезке на единицу.
- Проверить, есть ли среди точек с фиксированной четностью внутри отрезка хотя бы одна, значение в которой нулевое.

Так как значения в точках никогда не могут стать отрицательными, для таких операций подойдет дерево отрезком на минимум с прибавлением на отрезке. Для того, чтобы правильно обрабатывать точки нужной четности, можно использовать два дерева отрезков: одно для четных  $y$ -координат, другое для нечетных. Тогда каждая операция сводится либо к прибавлению на отрезке, либо к взятию минимума на определенном отрезке в каждом из деревьев.

Для того, чтобы получить полное решение, необходимо также применить метод сжатия координат, так как исходные координаты могут быть слишком большими для того, чтобы сохранить их

все в дереве отрезков. При сжатии координат нужно очень внимательно отнестись к тому, что нас иногда интересуют точки определенной четности. Это значит, что нас могут интересовать не только  $y$ -координаты всех вершин квадратов, но и другие  $y$ -координаты, расположенные между ними и имеющие нужную для нас четность. Самым простым способом решения этой проблемы является рассмотрение при сжатии координат не только всех  $y$ -координат квадратов, но и всех  $y$ -координат, отличающихся от них не более, чем на 2.

Двоичный поиск по ответу работает за время  $O(\log C)$ , где  $C = \max(r, c)$ . Проверка с помощью сканирующей прямой и дерева отрезков работает за время  $O(n \log n)$ . Таким образом, асимптотика времени работы всего решения составляет  $O(n \log n \log C)$ . Для получения полного балла по задаче, несмотря на небольшие ограничения, необходимо было довольно эффективно реализовать решение. Частичные баллы можно было набрать, не применяя сжатие координат, либо реализовав дерево отрезков «наивно».