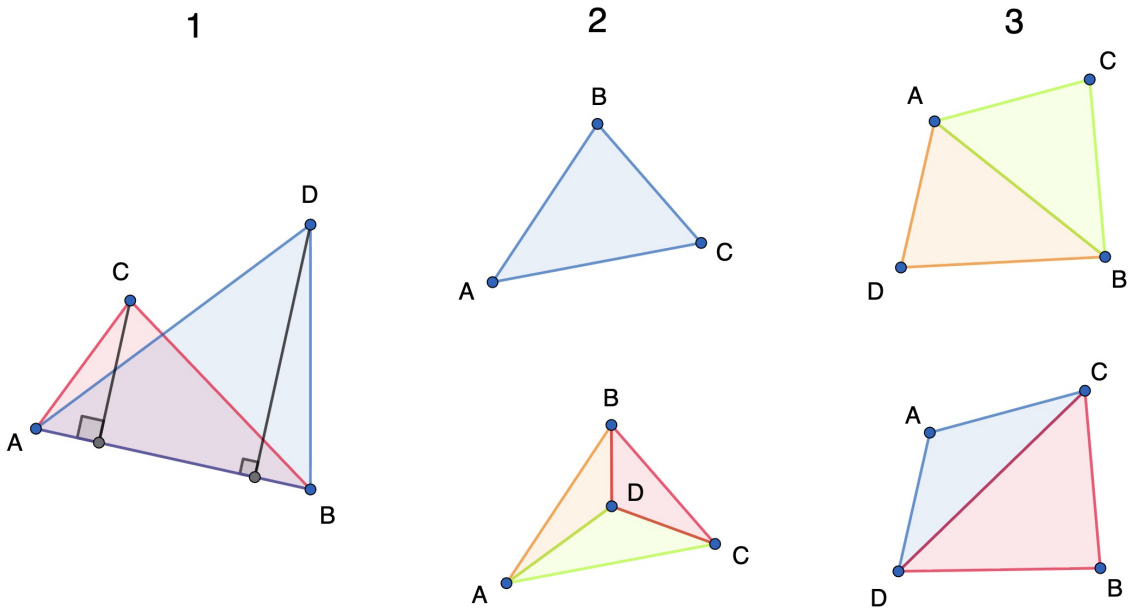
Разбор задачи «Разрезание торта»

Автор и разработчик задачи: Иван Сафонов

Заметим, что в задаче просят построить разбиение выпуклой оболочки множества точек на части. Для групп без максимизации достаточно найти выпуклую оболочку, для групп с максимизацией нужно найти триангуляцию множества точек.

Заметим, что с помощью нашего запроса мы можем делать следующие операции:

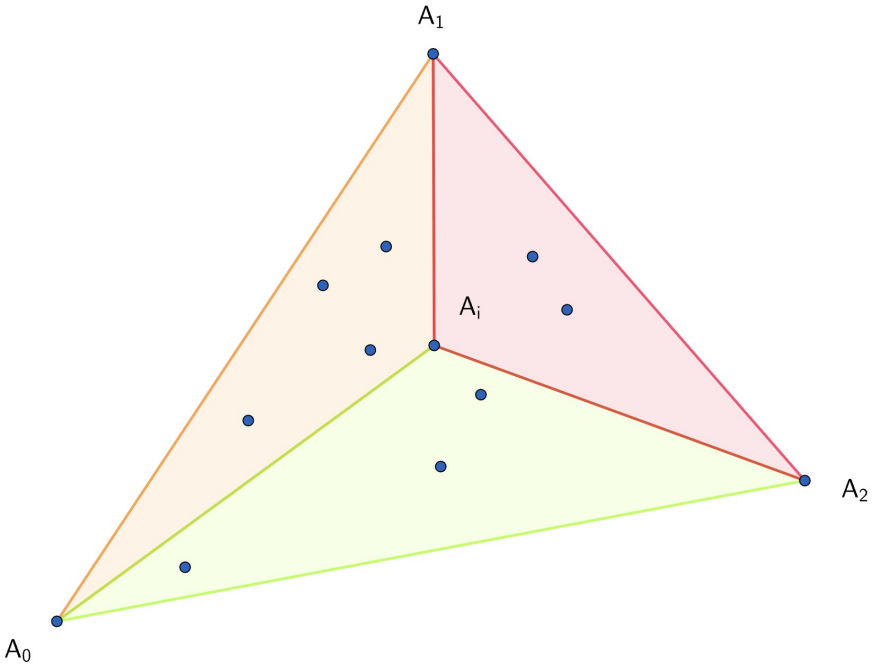
1. По данным точкам A, B, C, D проверить, какое из расстояний $\operatorname{dist}(C, AB)$, $\operatorname{dist}(D, AB)$ больше (расстояние до прямой). Для этого нужно просто сравнить $S(ABC)$ и $S(ABD)$.
2. По данным точкам A, B, C, D проверить, лежит ли точка D внутри треугольника ABC. Для этого нужно проверить, равны ли $S(ABC)$ и $S(ABD) + S(ACD) + S(BCD)$.
3. По данным точкам A, B, C, D проверить, пересекаются ли AB и CD. Для этого нужно проверить, равны ли $S(ABC)+S(ABD)$ и $S(ACD)+S(BCD)$. Именно в этом пункте мы пользуемся тем, что $ABCD$ не является трапецией, потому что для них возможно равенство без пересечения AB и CD.



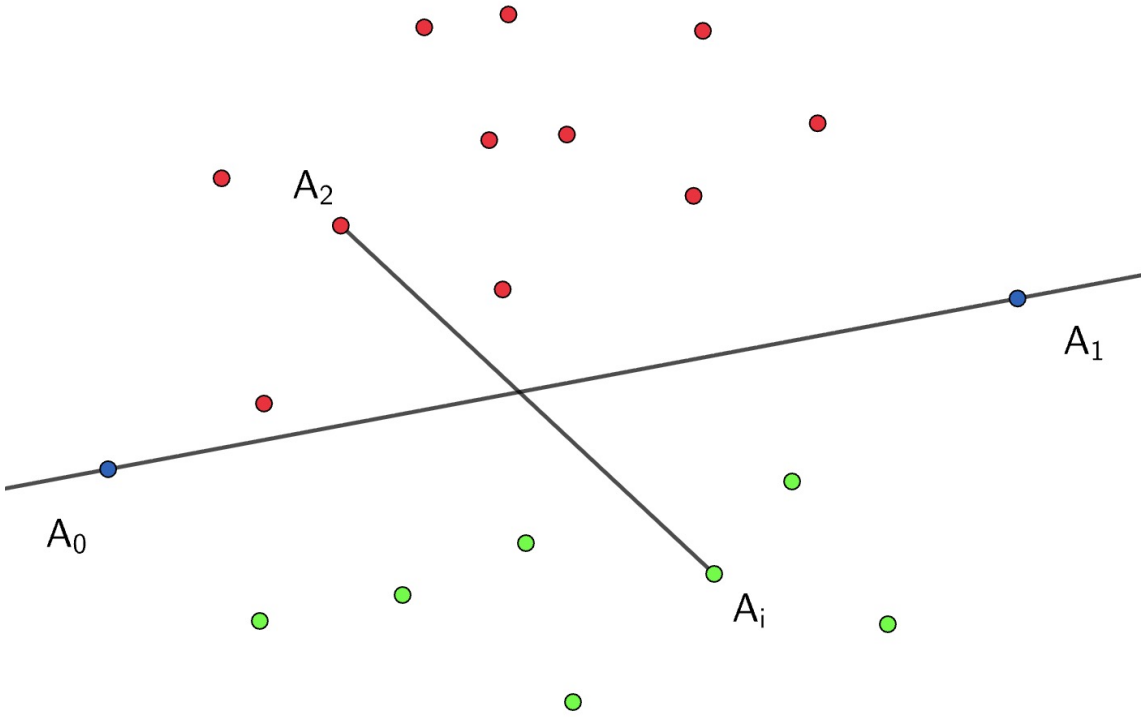
Чтобы решить 1 и 2 группы ($n = 4$) можно просто проверить, лежит ли каждая из точек внутри треугольника, образованного остальными точками. Если да, то триангуляция содержит 3 треугольника. Иначе точки образуют выпуклый четырехугольник. Его разбиение на 2 треугольника можно найти, например, с помощью проверки, какие два отрезка пересекаются (они будут диагоналями четырехугольника).

Заметим такую идею: можно найти точку, лежащую на выпуклой оболочке, за $n - 2$ запроса. Для этого возьмем две любых точки, например, A_0 и A_1 и среди всех остальных точек найдем треугольник с максимальной площадью $S(A_0A_1A_i)$. Тогда A_i точно на выпуклой оболочке, потому что расстояние $\operatorname{dist}(A_i, A_0A_1)$ максимально.

Таким образом можно найти три точки на выпуклой оболочке за $\leqslant 3n$ запросов. В 3 и 4 группе это будет вся выпуклая оболочка. Чтобы найти триангуляцию в этом треугольнике в группе 4 можно, например, выбрать случайную точку внутри, она разбивает треугольник на 3 меньших треугольника. Затем, с помощью запросов типа 2 можно для каждой из оставшихся точек найти, в какой из них она попадает, и решить задачу рекурсивно. Поскольку точки внутри треугольника случайные, это потребует $O(n \log n)$ запросов.



Чтобы решить следующие группы, в начале найдем две точки A_0, A_1 (Н.У.О. их индексы такие), которые лежат на выпуклой оболочке. Это потребует $\leqslant 2n$ запросов. Далее все оставшиеся точки можно разделить на два множества точек, лежащих в разных полуплоскостях относительно прямой A_0A_1. Для этого можно взять третью точку A_2 и каждую оставшуюся точку A_i проверять через проверку отрезков A_0A_1 и A_2A_i на пересечение. Это потребует еще $\leqslant n$ запросов. Далее будем решать задачу для каждой из полуплоскостей независимо. Поэтому теперь будем считать, что все точки лежат с одной стороны от прямой A_0A_1.



Отсортируем точки по расстоянию до прямой A_0A_1. Это потребует $\leqslant n \log_2 n$ запросов.

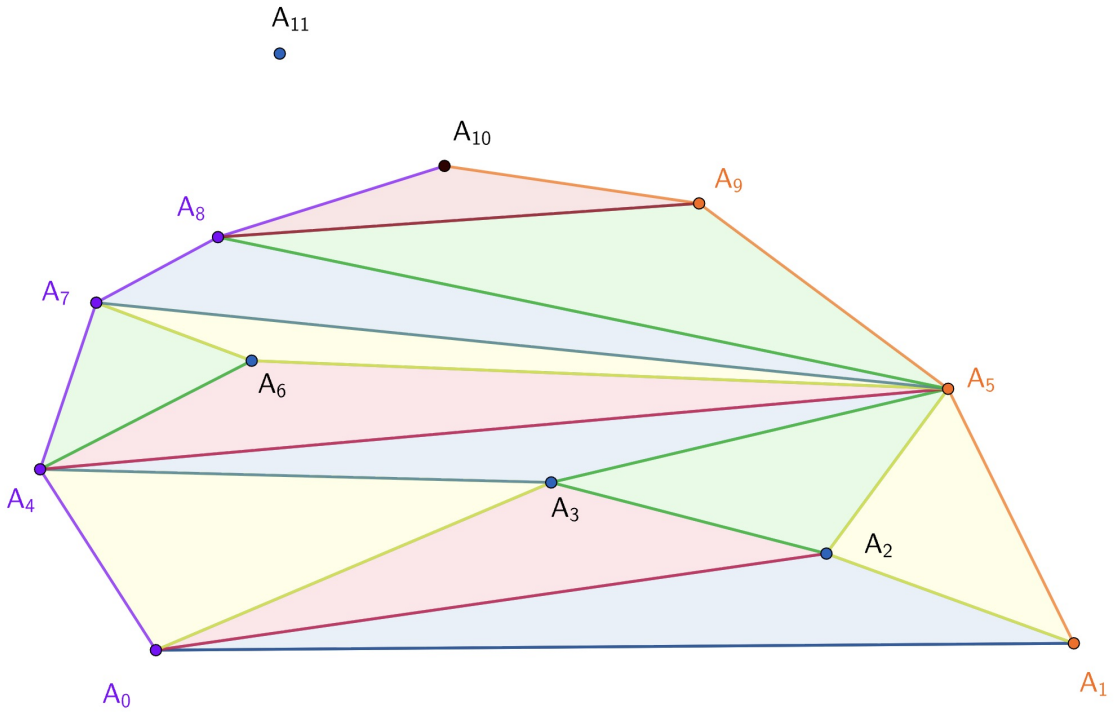
В группе 5 можно взять самую далекую точку A_i и снова поделить все остальные точки на лежащие по разные стороны от прямой A_0A_i за $\leqslant n$ запросов. Заметим, что теперь в каждой из

частей точки монотонны, и мы уже знаем их порядок по высоте. Таким образом, ответ будет найден за $\leqslant n \log_2 n + 4n$ запросов, что укладывается с запасом.

Обсудим полное решение. После сортировки по высоте применим алгоритм, похожий на алгоритм Грэхэма, но адаптированный для точек, отсортированных по расстоянию от прямой, а не по углу. Оказывается, что во время этого алгоритма мы также сможем строить триангуляцию.

Пусть точки отсортированы по увеличению расстояния от A_0A_1 в порядке $A_2, A_3, \ldots, A_{n-1}$. После обработки префикса этого порядка $A_2, A_3, \ldots, A_{i-1}$ будем поддерживать следующее состояние:

- Мы храним выпуклую оболочку точек $A_0, A_1, A_2, \ldots, A_{i-1}$ как объединение двух огибающих, назовем их левой и правой. Левая состоит из точек $[A_0, \ldots, A_{i-1}]$, правая состоит из точек $[A_1, \ldots, A_{i-1}]$.
- Мы строим триангуляцию этой выпуклой оболочки.

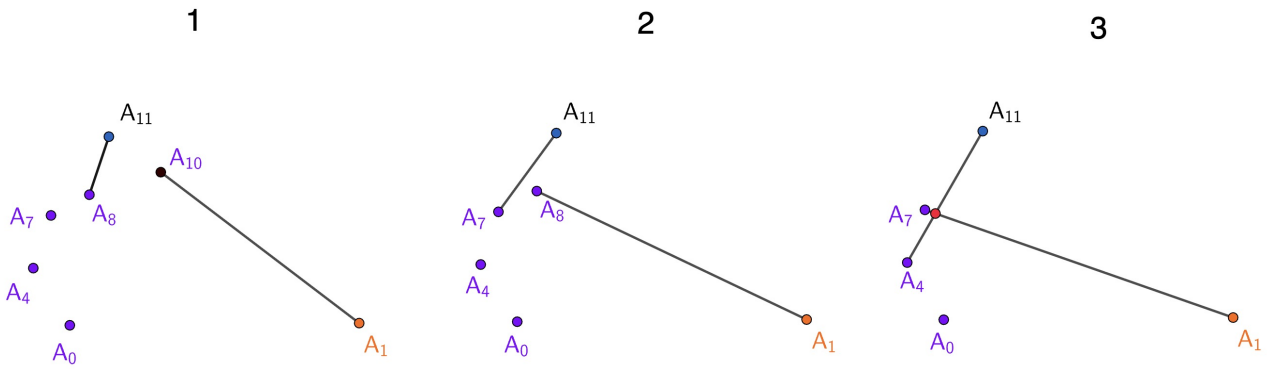


В данном примере $i = 11$. Левая огибающая состоит из точек $[A_0, A_4, A_7, A_8, A_{10}]$, правая огибающая состоит из точек $[A_1, A_5, A_9, A_{10}]$. Мы добавляем точку A_{11}.

Таким образом, в конце мы найдем ответ. Для инициализации (единственной точки A_2) построим левую огибающую как $[A_0, A_2]$, правую как $[A_1, A_2]$. Триангуляция будет содержать единственный треугольник $A_0A_1A_2$.

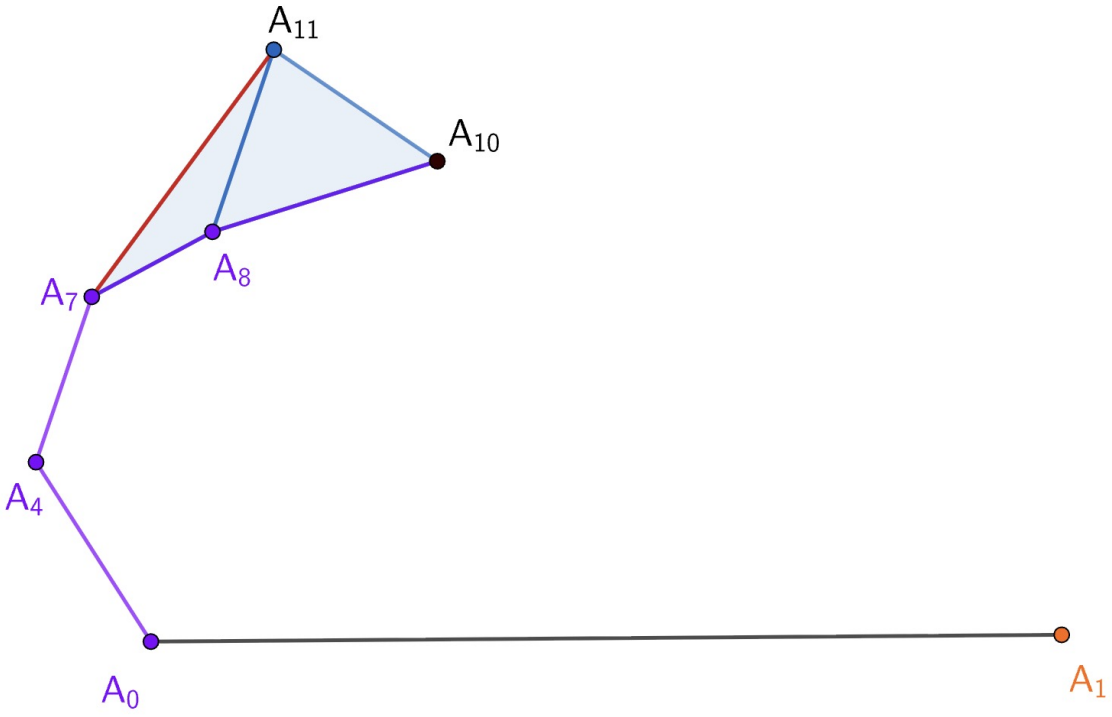
Далее научимся добавлять новую точку A_i в эту конструкцию. Заметим, что нам нужно независимо добавить эту точку в левую и правую огибающие.

Пусть левая огибающая это $[A_{l_0}, A_{l_1}, \ldots, A_{l_k}]$, где $l_0 = 0$, $l_k = i - 1$. Заметим, что сейчас нам нужно, как и в обычном алгоритме Грэхэма, удалить некоторый суффикс огибающей и добавить в конец A_i. Чтобы удалить нужный суффикс, мы должны брать две последние точки A_{l_j}, $A_{l_{j+1}}$ и удалять последнюю точку $A_{l_{j+1}}$, если $\overrightarrow{A_{l_j}A_{l_{j+1}}} \times \overrightarrow{A_{l_{j+1}}A_i} > 0$ (векторное произведение). Заметим, что это равносильно тому, что отрезки $A_{l_j}A_i$ и $A_{l_{j+1}}A_1$ не пересекаются, что мы умеем проверять с помощью запроса типа 3. Тогда давайте удалим нужный суффикс огибающей с помощью таких проверок.



При добавлении A_{11} в левую огибающую, мы будем делать следующие шаги: пересекаются ли A_8A_{11} и $A_{10}A_1$? нет, поэтому удаляем A_{10}; пересекаются ли A_7A_{11} и A_8A_1? нет, поэтому удаляем A_8; пересекаются ли A_4A_{11} и A_7A_1? да, поэтому останавливаемся и добавляем A_{11} в конец.

Более того, давайте при каждом удалении последней точки $A_{l_{j+1}}$ из огибающей добавлять в триангуляцию треугольник $A_{l_j}A_{l_{j+1}}A_i$. Тогда такими треугольниками мы в точности покроем образованную часть между касательной от точки A_i и старой огибающей.



При обновлении левой огибающей мы добавляем треугольники $A_8A_{10}A_{11}$ и $A_7A_8A_{11}$ в триангуляцию.

Такой же алгоритм нужно повторить для правой огибающей, но используя точку A_0 в проверках вместо A_1.

Таким образом, мы построим выпуклую оболочку и ее триангуляцию. Давайте заметим, что этот алгоритм потребует $\leqslant 2n$ запросов для одной огибающей, потому что каждая точка может быть добавлена и удалена в огибающую не более одного раза. Таким образом, эта часть решения займет $\leqslant 4n$ запросов.

Таким образом, все решение требует $\leqslant n\log_2 n+7n$ запросов. Если чуть более аккуратно оценить реальную константу сортировки при использовании merge sort, можно заметить, что такое решение укладывается в $20n$ запросов при $n \leqslant 10^4$. Заметим, что оценку сортировки можно сократить еще чуть-чуть, если для сортировки использовать алгоритм, который тратит $\leqslant \sum\limits_{i=1}^{n}\lceil\log_2 i\rceil$ запросов, за счет того, что n раз вставляет новый индекс в отсортированный порядок с помощью бинпоиска.

Разбор задачи «Марш овец»

Автор задачи: Александр Бабин
Разработчик задачи: Юрий Федоров

1 $T = 2$

Заметим, что каждую секунду четность позиции каждой овцы меняется. Поэтому, если существует две овцы с разной четностью позиции, то ответ, очевидно `No`, так как всегда будет овца как на четной, так и нечетной позиции. В противном случае утверждается, что ответ всегда существует. Например, существует следующий алгоритм:

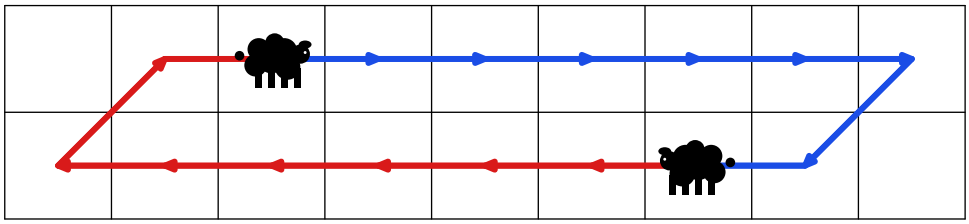
Если у текущего последнего столбца четность не совпадает с четностью позиций овец, то всегда можно обрезать участок на 1 (так как на последнем столбце гарантированно не может быть овцы, ввиду четности). Таким образом можно достичь количества столбцов равному 1, где все овцы гарантированно двигаются одинаково.

2 $n \leqslant 3, m \leqslant 4$

Если решить $T = 1$, то оставшихся случаев очень мало и их можно рассмотреть руками, либо же можно сделать перебор. В качестве состояния можно взять позиции каждой из овец, а также длину участка. И тогда есть два вида переходов: мы симмулируем одну секунду, либо же уменьшаем участок. Что можно реализовать при помощи любого графового алгоритма. Также существуют другие переборы.

3 $n = 2$

В случае $a_1 = 1, a_2 = m$ можно порисовать и понять, что ответ равен $\frac{m+1}{2}$, и достаточно обрезать до этого размера через это же время.



Теперь давайте посмотрим на два расстояния, отмеченных синим и красным цветом. Это расстояние от первой овцы до второй, и от второй до первой в порядке их пути. Тогда заметим, два факта:

1. Овцы будут двигаться одиннаково, тогда и только тогда, когда расстояние от одной до другой будет равно строго нулю.

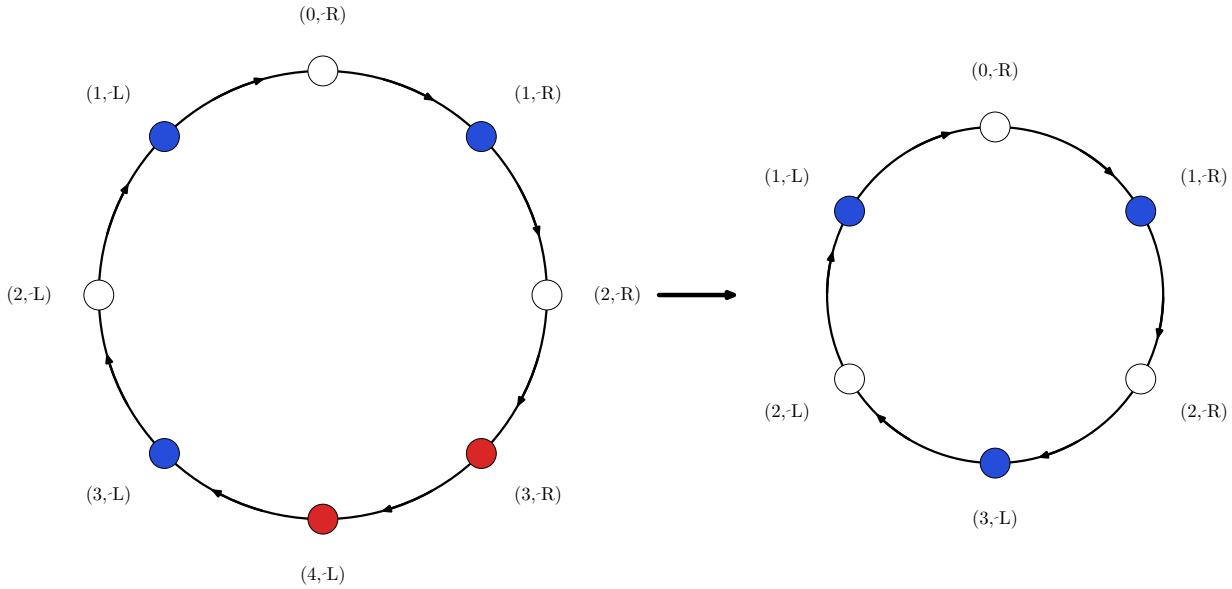
2. После уменьшения размера поля на 1, одно из этих расстояний уменьшается на 2.

Из чего сразу следует решение за $O(m)$: мы можем симмулировать их движение, и если обрезание уменьшает наименьшее из расстояний, то мы будет обрезать участок на 1. И за $4 \cdot m$ операций этот процесс гарантированно будет завершен.

Теперь же научимся решать за $O(1)$: заметим, что если мы оставляем максимальное из расстояний, то если эта длина равна x, то ответ будет равен $\frac{x}{2} + 1$, так как если мы оставили длину x, то расстояние от одной овцы до другой будет равно $2 \cdot x - 2$. Тонкости же восстановления ответа, будут рассмотрены в полном решении.

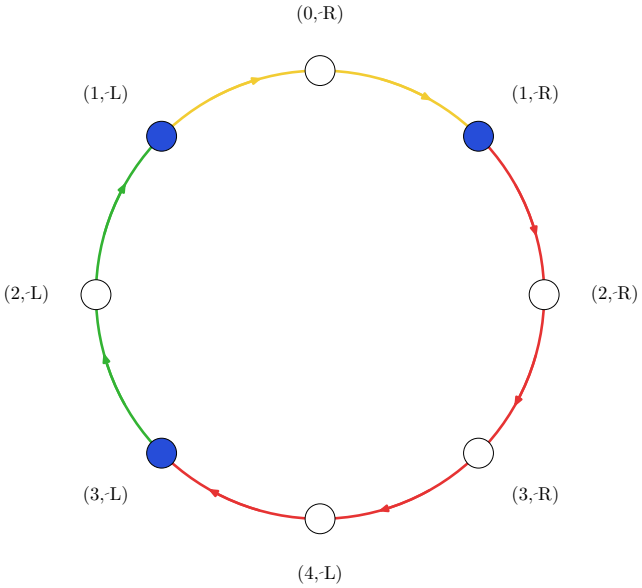
4 $T = 2$

Немного разовьем идеи из $n = 2$. Заметим, что участок можно представить в виде цикла длины $2 \cdot m - 2$. А уменьшение длины участка на 1 — это уменьшение длины цикла на 2:



На этой картинке приведен пример, где 3 овцы находятся в позициях $(2, 4, 2)$, их направления движения `RLL` (то есть синие точки — это овцы). А для удобства дальнейшего повествования позиции перенесены в 0 индексации. Красными вершинами помечены те, которые удаляются после уменьшения цикла.

Тогда расстояния до соседних овец можно представить в виде количества дуг от овцы до следующей по ней по циклу (здесь разными цветами обозначены различные дуги, которые представляют собой расстояния до соседних овец):



Тогда заметим, что конструкция никак принципиально не отличается от ситуации $n = 2$. Нас также интересует только максимальное расстояние между соседними овцами на цикле. И можно, как и версии $n = 2, m \leqslant 2 \cdot 10^5$ просимулировать движения и обрезать в моменты, когда текущая длина не является максимумом, в этом случае получается асимптотика $O(n \cdot m)$. Либо же сказать, что ответ равен $\frac{x}{2} + 1$, где x — максимальное расстояние между соседними овцами.

5 Полное решение

Осталось только научиться восстанавливать. В случае $n, m \leqslant 1000$ мы уже научились решать, поэтому осталось научиться восстанавливать при полных ограничениях.

Давайте отсортируем всех овец по расположению по циклу. И пусть i — это позиция овцы такая, что дуга от овцы i до следующей овцы по циклу максимальное. Тогда восстановить можно следующим образом:

- Перенумеруем массив таким образом, чтобы овца под номером i оказалась самой последней овцой.
- Давайте подождем время, чтобы последняя овца оказалась на позиции $m-1$ на круге в 0 индексации (или другими словами, чтобы она стояла в последнем столбце).
- Тогда чтобы обрезать дугу от овцы $n-1$ до овцы n, достаточно будет «прокрутить» цикл, чтобы овцы стала на позицию $m-1+\frac{d}{2}$, где d — длина дуги от овцы $n-1$ до овцы n. А затем уменьшить длину участка на $\frac{d}{2}$, после чего овца под номером будет n будет стоять на позиции $m-1$ в цикле.
- После этого мы можем игнорировать овцу под номером n и обрабатывать овцу под номером $n-1$ и так далее.

Ну и несложно заметить, что это действительно удалит все дуги, кроме максимальной.

Разбор задачи «Украшение»

Автор задачи: Алексей Михненко
Разработчик задачи: Виктор Романенко

Формальная постановка задачи

Дано n отрезков на прямой. Разрешена следующая операция: выбрать два отрезка (l_i, r_i) и (l_j, r_j) и произвольным образом составить два новых отрезка из их четырёх границ.

Дано q запросов k_i. Для каждого запроса требуется определить минимальное количество операций, необходимых для того, чтобы существовало подмножество из k_i попарно пересекающихся отрезков.

Основная идея

Пусть все отрезки искомого множества пересекаются в некоторой точке x. Предположим, что:

- c — число отрезков, уже покрывающих точку x,
- l — число отрезков, расположенных строго левее x,
- r — число отрезков, расположенных строго правее x.

Заметим, что одной операцией можно взять один левый и один правый отрезок и перестроить их так, чтобы оба пересекали точку x. Следовательно, за одну операцию можно увеличить число отрезков, пересекающих x, на два.

Таким образом, максимальное количество отрезков, которые можно сделать пересекающимися в точке x, равно

$$c + 2 \cdot \min(l, r).$$

Для достижения этого значения потребуется $\min(l, r)$ операций. Если требуется получить промежуточное количество k_i, то достаточно выполнить

$$\left\lceil \frac{k_i - c}{2} \right\rceil$$

операций.

Подгруппа 3. Отрезки не пересекаются

В этом случае для любой точки выполняется $c \leqslant 1$. Поэтому удобно рассматривать точку внутри центрального отрезка.

Для любой точки внутри такого отрезка выполняется $c = 1$, следовательно ответ для любого k_i вычисляется по формуле

$$\left\lceil \frac{k_i - 1}{2} \right\rceil.$$

Если число отрезков чётное и $k_i = n$, то точка пересечения должна находиться между двумя центральными отрезками, где $c = 0$. Тогда ответ равен $n/2$. Поскольку n чётно, это значение эквивалентно формуле выше.

Подгруппа 4. $k_i = n$

Если требуется сделать пересекающимися все отрезки, точка ответа должна находиться в позиции, где количество отрезков строго слева и строго справа одинаково.

Такую точку можно найти с помощью сканлайна. После нахождения соответствующих значений c, l и r ответ вычисляется по приведённой ранее формуле.

Решение подгрупп 1, 2, 4, 5

Выполним сканлайн. Для каждой открывающей границы отрезка и каждой закрывающей границы будем фиксировать значения:

- c — количество отрезков, покрывающих текущую точку,
- $\min(l, r)$ — минимальное число отрезков строго слева и строго справа.

При обработке закрывающей границы соответствующий отрезок уже не учитывается в c, однако считается лежащим строго слева, поскольку точка пересечения может находиться вне его границ.

Для каждой такой позиции можно вычислить минимальное число операций по формуле

$$\left\lceil \frac{k_i - c}{2} \right\rceil,$$

при условии, что

$$k_i \leqslant c + 2 \cdot \min(l, r).$$

Минимум по всем таким позициям даёт ответ для подгрупп 1, 2, 4 и 5.

Полное решение

Остаётся оптимизировать вычисление ответов.

Для каждого k от 1 до n заранее вычислим максимальное значение c, такое что возможно получить k пересекающихся отрезков. Обозначим эту величину через $best[k]$.

Во время сканлайна для каждой позиции вычисляется значение

$$c + 2 \cdot \min(l, r),$$

которое показывает максимальное число отрезков, пересекающихся в данной точке после операций. Тогда выполняем обновление

$$best[c + 2 \cdot \min(l, r)] = c.$$

После завершения сканлайна пройдёмся по массиву $best$ справа налево и распространим максимум:

$$best[i] = \max(best[i], best[i + 1]).$$

Теперь $best[k]$ содержит максимальное число отрезков, уже покрывающих точку, из которой можно получить k пересекающихся отрезков.

Ответ на запрос k_i вычисляется по формуле

$$\left\lceil \frac{k_i - best[k_i]}{2} \right\rceil.$$

Итого $O(n \log n)$ на предпосчет из-за сортировки и $O(1)$ на запрос

Разбор задачи «Монстры и мечи»

Автор и разработчик задачи: Ренат Каримов

Подгруппа $k = 1$

Для решения подгруппы k=1 введем сначала новое обозначение: w_i = минимальная стоимость меча, при помощи которого можно убить i-го монстра. Если же для какого-то монстра такого меча нет, то ответ сразу NO.

В этой подгруппе мы должны убивать монстров подряд по одному, следовательно каждый раз нужно находить стоимость самого дешёвого меча, который может убить очередного монстра. Это и есть w_i. Тогда чтобы решить задачу нам нужно просто n раз проверить, что количество монет, которое у нас осталось больше или равно очередного w_i. Решение работает $O(n \log n + m \log m)$

Подгруппа $k = n$

Для решения подгруппы с $k = n$ нужно заметить, что при покупке нового меча нам всегда нужно выбирать меч с сильной больше, чем предыдущего. Это так потому что у нас нет ограничения на прочность меча. Тогда из этого факта следует замечание, что в оптимальном решении мы всегда можем использовать меч до упора. То есть нужно менять меч только в ситуации, когда он не может убить очередного монстра.

Используя эти факты можно написать решение с использованием техники динамическое программирование. Обозначим $dp[i]$ = максимальное количество монет, которые может остаться у рыцаря после убийства первых i монстров (при этом нам не важно, какими мечами мы убивали монстров).

Для удобства обозначим $w(i, j) = \max\limits_{i \leqslant l \leqslant j} w[l]$. Тогда изначальное состояние: $dp[0] = x$ и переход $dp[i] = \max\limits_{j} dp[j] - w(i, j-1) + r(i, j-1)$. При этом должен быть верно, что $dp[j] \geqslant w(i, j-1)$.

Для оптимизации такой дпшки можно заметить факт, что нас интересуют не все состояния этой дпшки. Для каждого меча можно выделить префикс монстров, на котором он может всех победить. Тогда на основе фактов выше можно пересчитывать дпшки только через эти префиксы, так мы получим решение за $O(m^2 + n \log n)$.

Для полного решения можно нужно просто оптимизировать динамику выше, это можно сделать, например, при помощи структуры декартово дерево, тогда итоговое решение будет работать за $O(n \log n + m \log m)$

Подгруппа k - произвольное

Для решения подгруппы k - произвольное нужно подробнее рассмотреть получившуюся формулу. Пусть, значение i - фиксированно, тогда уменьшая j значение $w(i, j)$ будет увеличиваться. Из-за этого хочется использовать технику стека максимумов. При помощи нее можно реализовать такую же дпшку как и для случая k=n только ещё с условием, что $i - j \leqslant k$. Так можно получить решения за $O(n \log n + m \log m)$ при помощи ДД, ХЛД или прохода с сетом.

Однако подробнее мы разберем немного другую дпшку. Обозначим $dp[i]$ = минимальное количество золота, которое нам нужно, чтобы гарантированно убить всех монстров с номерами i ... n-1 (в порядке возрастания номеров). Тогда ответ YES только в случае, если $dp[0] \leqslant x$.

Изначально значение: $dp[n-1] = w[n-1]$. Переход: $dp[i] = \min\limits_{i < j \leqslant i+k} (w(i, j-1) + \max(0, dp[j] - r(i, j-1)))$. Обозначим $pr[i] = r(0, i)$ и немного перепишем получившуюся формулу, получим $w(i, j-1) + max(0, dp[j] - pr[j-1] + pr[i])$. Тогда нам интересен знак $dp[j] - pr[j-1] + pr[i]$. При этом несложно заметить, что если теперь уже j - фиксированно, а i уменьшается, то значение $dp[j] - pr[j-1] + pr[i]$ только убывает, следовательно знак меняется не более одного раза.

Такое дп можно быстро пересчитывать при помощи стека максимумов и сетов, заметив факт, что $dp[i] - pr[i-1] \geqslant dp[i+1] - pr[i]$.

Итоговое решение будем иметь ассимптотику $O((n + m) \log (n + m))$