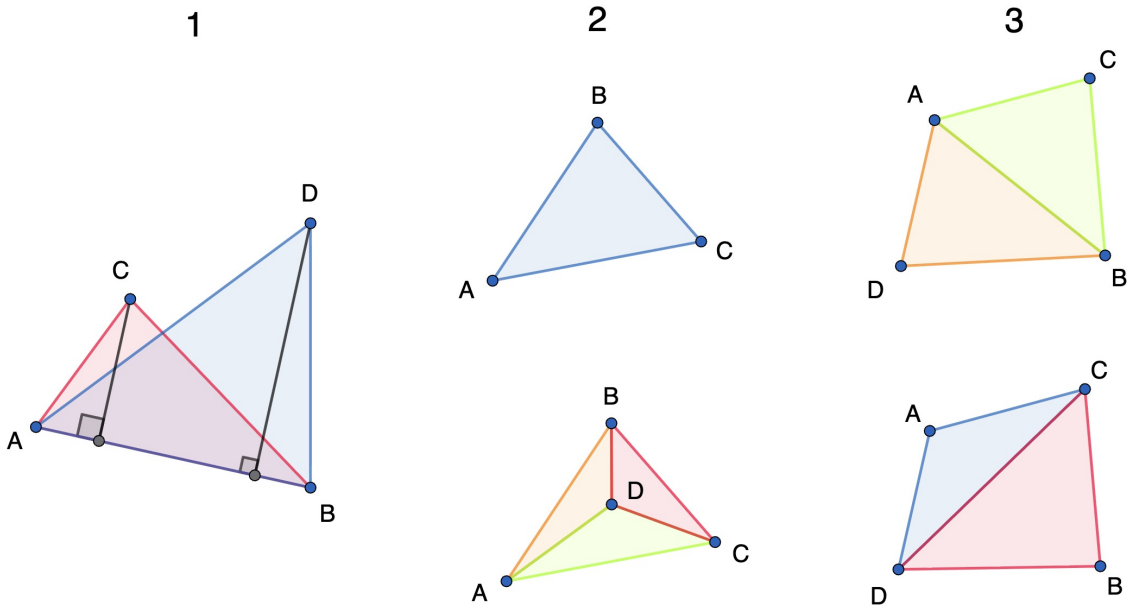
Разрезание торта

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:

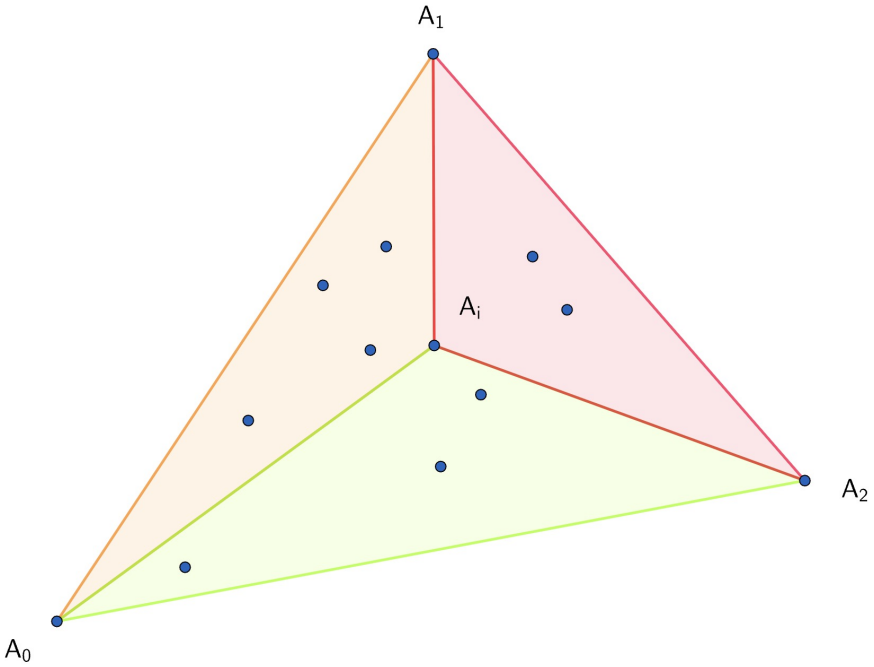
1. По данным точкам A, B, C, D проверить, какое из расстояний $\mathrm{dist}(C, AB)$, $\mathrm{dist}(D, AB)$ больше (расстояние до прямой). Для этого нужно просто сравнить $S(ABC)$ и $S(ABD)$.
2. По данным точкам A, B, C, D проверить, лежит ли точка D внутри треугольника ABC. Для этого нужно проверить, 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.



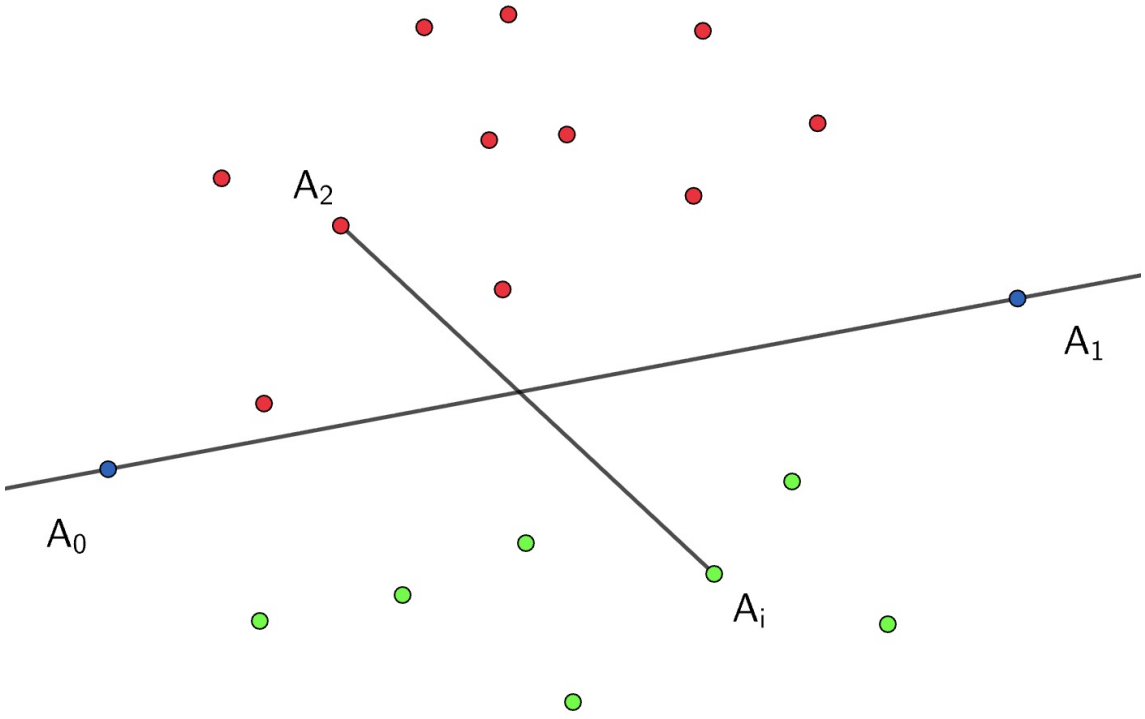
Чтобы решить 1 и 2 группы ($n = 4$) можно просто проверить, лежит ли каждая из точек внутри треугольника, образованного остальными точками. Если да, то триангуляция содержит 3 треугольника. Иначе точки образуют выпуклый четырехугольник. Его разбиение на 2 треугольника можно найти, например, с помощью 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 $\mathrm{dist}(A_i, A_0A_1)$ 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.

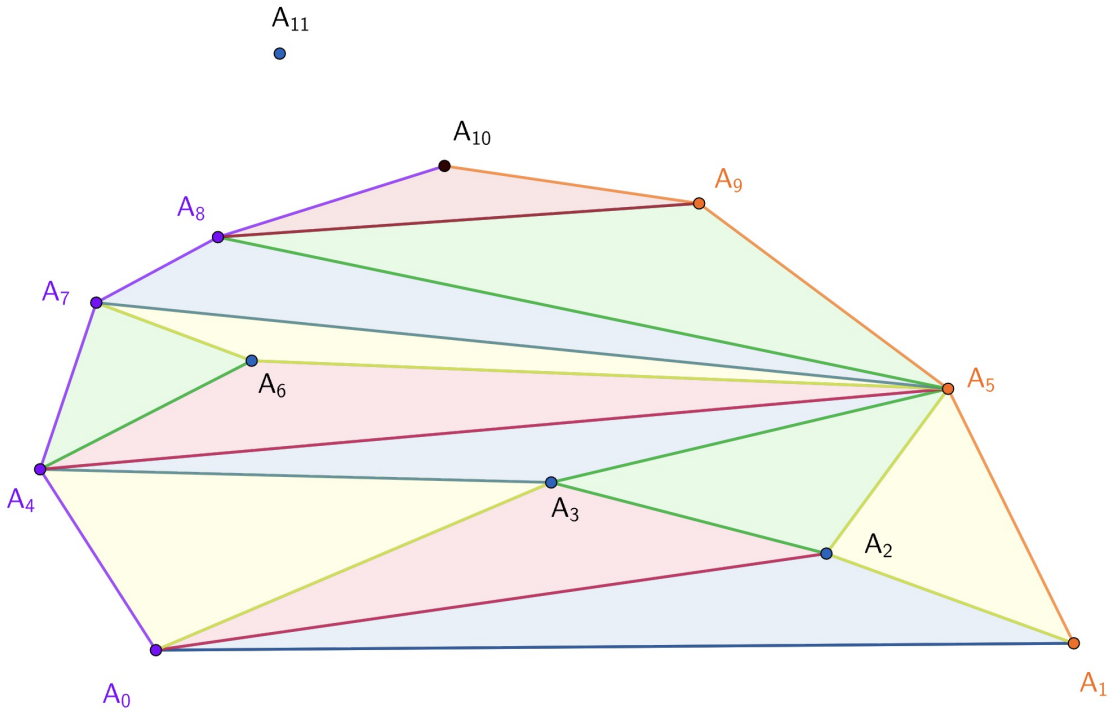
В группе 5 можно взять самую далекую точку A_i и снова поделить все остальные точки на лежащие по разные стороны от прямой A_0A_i за $\leqslant n$ запросов. Заметим, что теперь в каждой из

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.

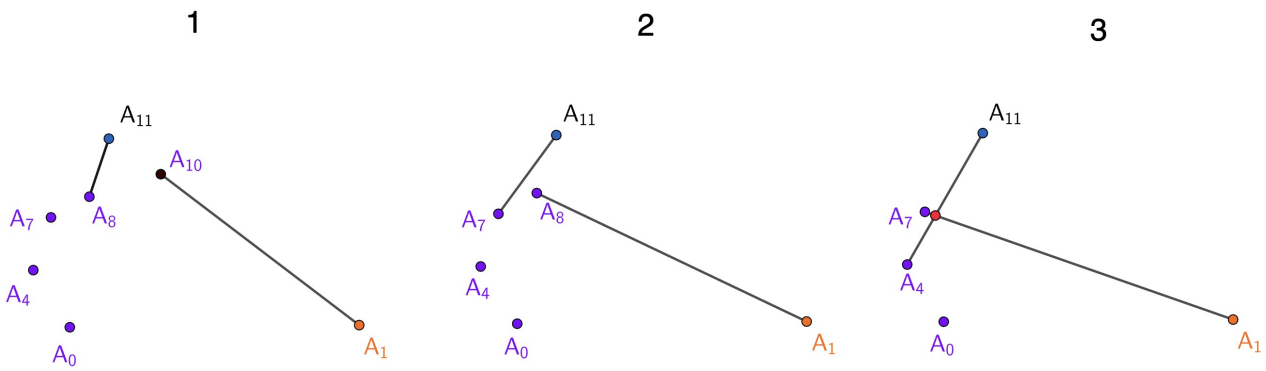


В данном примере $i = 11$. Левая огибающая состоит из точек $[A_0, A_4, A_7, A_8, A_{10}]$, правая огибающая состоит из точек $[A_1, A_5, A_9, A_{10}]$. Мы добавляем точку A_{11}.

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 $A_0A_1A_2$.

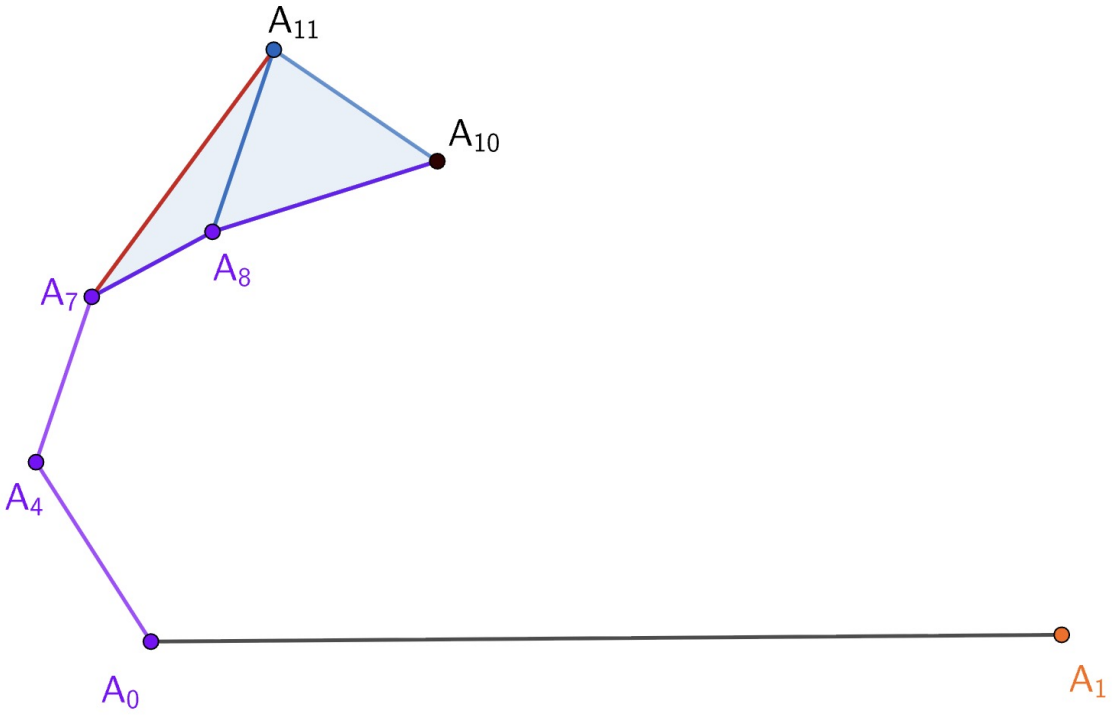
Далее научимся добавлять новую точку A_i в эту конструкцию. Заметим, что нам нужно независимо добавить эту точку в левую и правую огибающие.

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.