У нас есть набор точек и мы хотим найти минимальный выпуклый многоугольник, у которого все заданные точки лежат либо внутри, либо на границе.

Все вершины получившегося многоугольника должны принадлежать исходному множеству точек.

Из набора точек можно построить разные многоугольники, но выпуклый — только один.

Важно также понять, зачем нам нужна выпуклая оболочка.

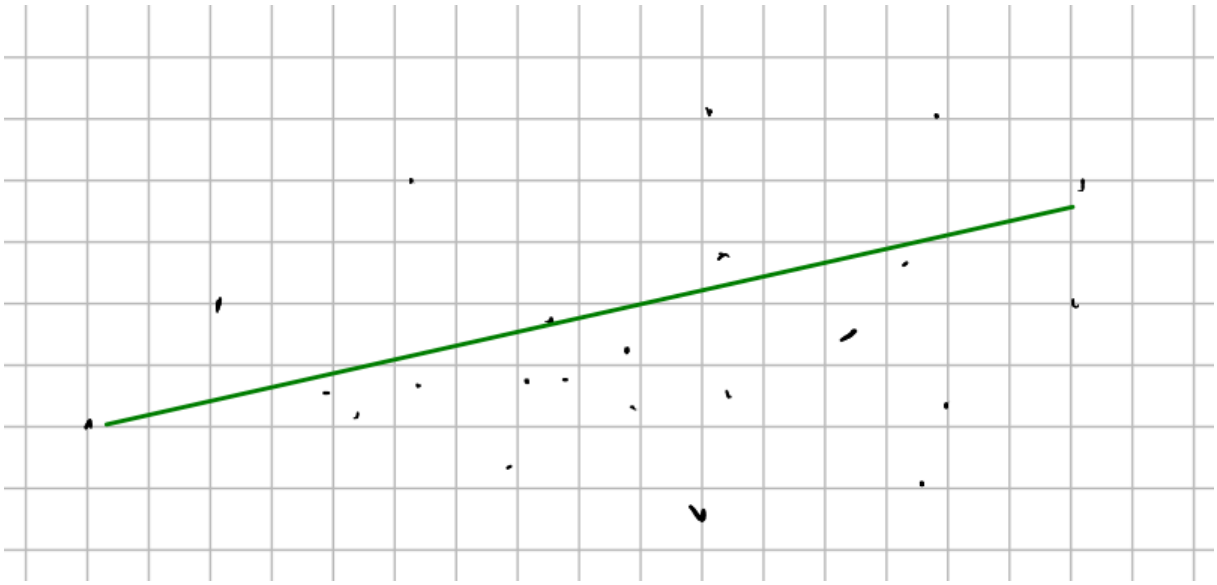
Алгоритм построения выпуклой оболочки

Как определять для двух точек, что они будут на границе, внутри или вершинами, а также в каком порядке?

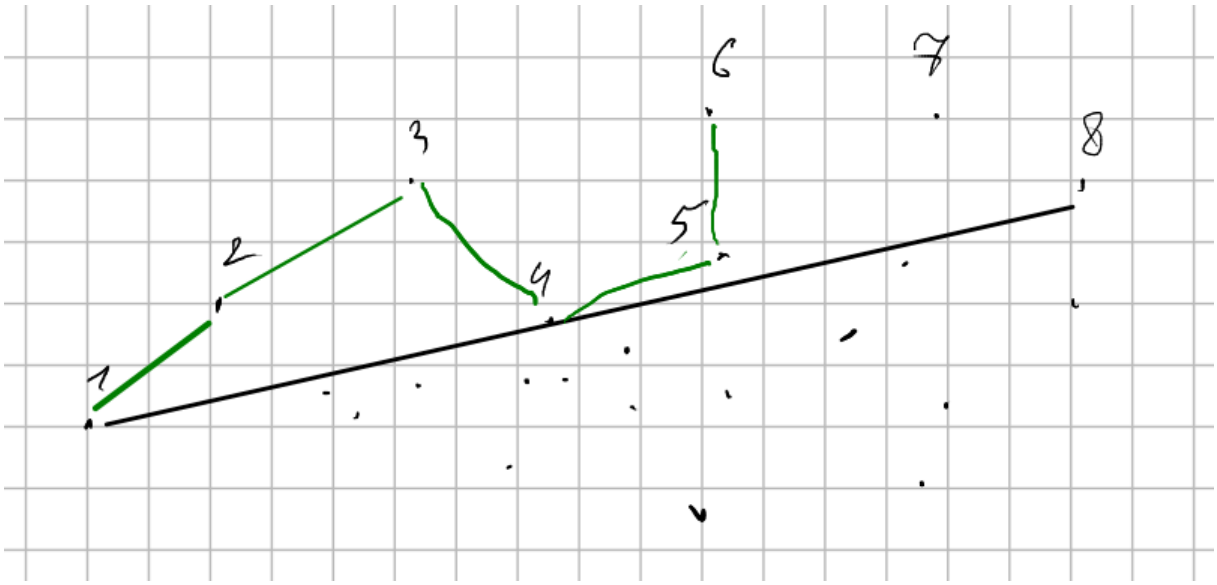
Рекомендуется использовать алгоритм верхнего и нижнего огибающего:

для каждого многоугольника мы можем провести вертикальные линии вниз и вертикальные линии вверх. Таким образом мы с лёгкостью определим верхнюю и нижнюю стороны многоугольника.

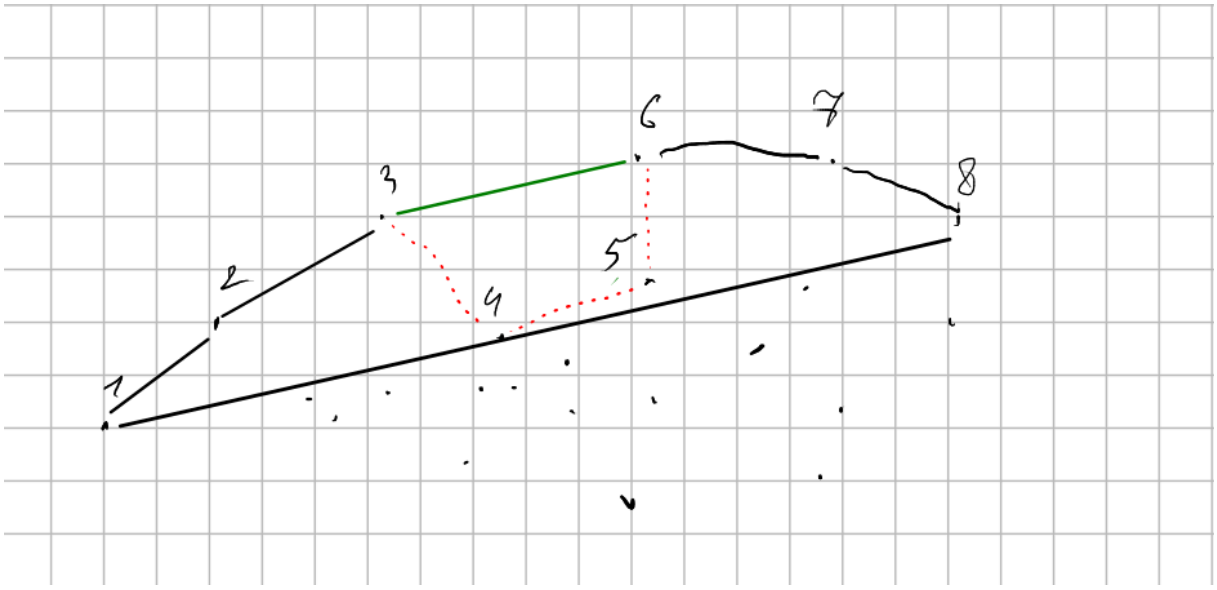
1. Соединим крайние левые и правые точки — это будет граница между верхней и нижней частями.



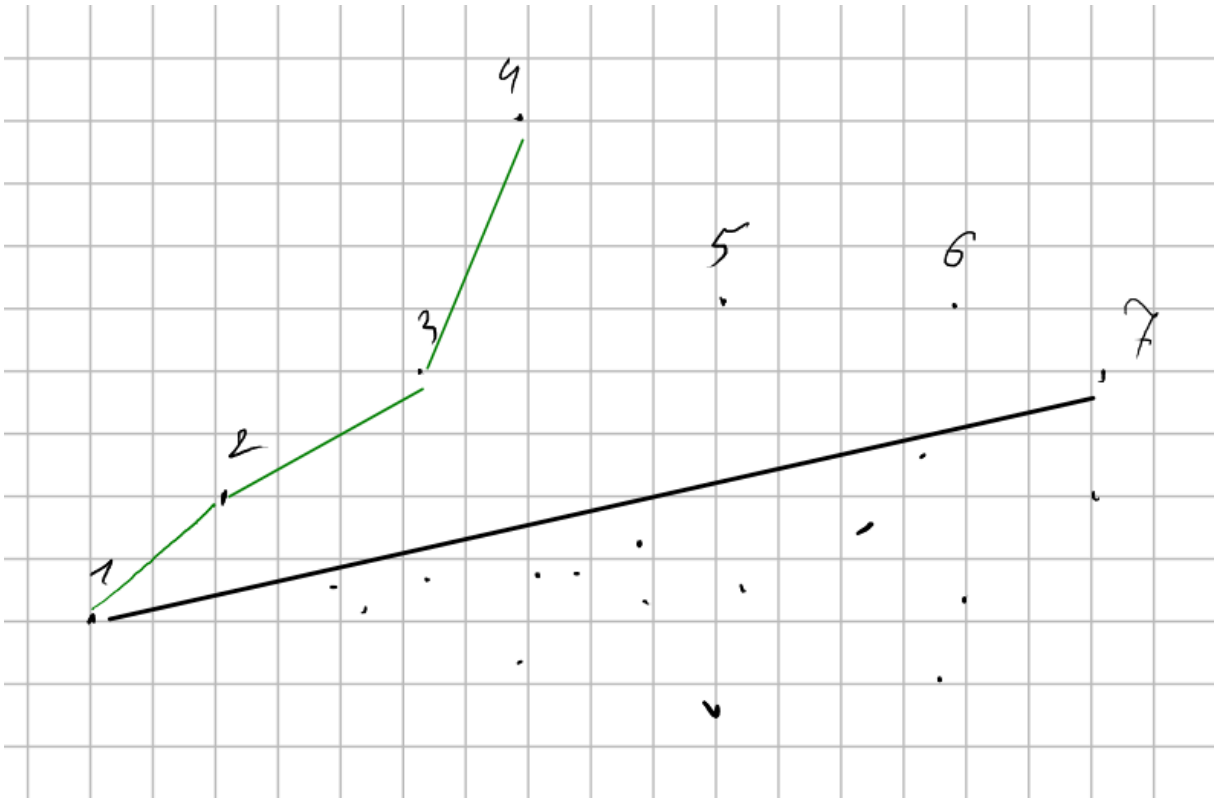
2. Попробуем соединить эти линии



3. Способ 1: Мы должны заметить, что 4 и 5 ниже точек 4 и 6. По этой причине исключим их.
3. Способ 2: Угол в многоугольнике от нуля до 180 градусов (не включая 180). Когда многоугольник становится невыпуклым, появляется тупой угол. Синус угла мы считать не хотим, поэтому мы будем вычислять псевдоскалярное произведение. Если знак положительный, то всё хорошо. Иначе всё плохо и выкидываем лишнюю точку.



Но может возникнуть ситуация другого рода:



1. Идём по точкам слева направо
2. Добавление новой точки заставляет нас выкинуть предыдущую? Выкидываем и проверяем новую предыдущую. И так пока ситуация не разрешится.

За сколько это работает? Самая тяжёлая часть — отсортировать точки слева направо, а основной алгоритм работает за линию.

Для хранения лучше всего использовать вектор, потому что через него проще смотреть как на конец, так и на соседей. В двусвязном списке тоже можно, но тут вопрос рациональности — вектором тоже можно.

Решения задач

Выпуклая оболочка

```
#include <bits/stdc++.h>

using namespace std;


bool operator<(const vec2d& a, const vec2d& b) {
  if(a.x != b.x) return a.x < b.x;
  return a.y < b.y;
}

void add(Vector<vec2d> v, vec2d p, bool upper) {
  // У нас есть вектор точек и мы добавляем в него какую-то точку p.
  // Нам нужно проверить и выкинуть предыдущие точки, если они плохие.
  // При этом, если в огибающем множестве меньше двух точек, то нам и выкидывать-то
нечего.
  while(v.size() >= 2) {
    // Взяли последнюю и предпоследнюю точки
```

```
    auto last = v.back();
    auto prelast = v[v.size() - 2];
    // Для верхнего огибающего нам нужно проверить,что наша верхняя точка p
выкидывает точку last
    vec2d lastp = p - last;
    vec2d prelastp = p - prelast;
    long long prod = prelastp % lastp; // % - оператор псевдоскалярного произведения,
он не коммутативен
    // в зависимости от того, в какой мы огибающей и выше или ниже предыдущая точка,
принимаем решение
    if((upper && prod >= 0) || (!upper && prod <= 0)) {
      // либо удаляем предыдущую точку, если она делает нашу фигуру выпуклой
      v.pop_back();
    } else {
      // либо останавливаем цикл
      break;
    }
    // вставляем точку
    v.push_back(p);
  }
}

vector<vec2d> convex_hull(vector<vec2d> points) {
  sort(points.begin(), points.end());
  vec2d A = points[0];
  vec2d B = points.back();
  vector<vec2d> upper = { A }; // верхняя огибающая
  vector<vec2d> lower = { A }; // нижняя огибающая
  for(int i = 1; i < points.size(); ++i) {
    // в целом мы можем добавить каждую точку в обе огибающие, ведь
    // алгритм всё равно выкинет ненужные точки --- зачем дважды выполнять одну
работу?
    add(upper, points[i], true);
    add(lower, points[i], false);
  }
  // осталось спаять огибающие в один многоугольник
  // удалим лишние точки в lower
  lower.pop_back();
  lower.erase(lower.begin());
  // обратим внимание, что в верхнем огибающем точки идут по часовой, а в нижнем ---
против
  // это не окей, ревёрснем нижнюю огибающую
  reverse(lower.begin(), lower.end());
}


int main() {
  cout << setprecision(9) << fixed;
  int n;
  cin >> n;
  vector<vec2d> polygon(n);
  for(int i = 0; i < n; ++i) cin >> polygon[u].x >> polygon[i].y;
  polygon = convex_hull(polygon);
  cout << polygon.size() << '\n';
  for(vec2d point : polygon) cout << point.x << ' ' << point.y << '\n';
}
```

605C Мечты фрилансера

Самая красивая задача на выпуклую оболочку — “Мечты фрилансера”.

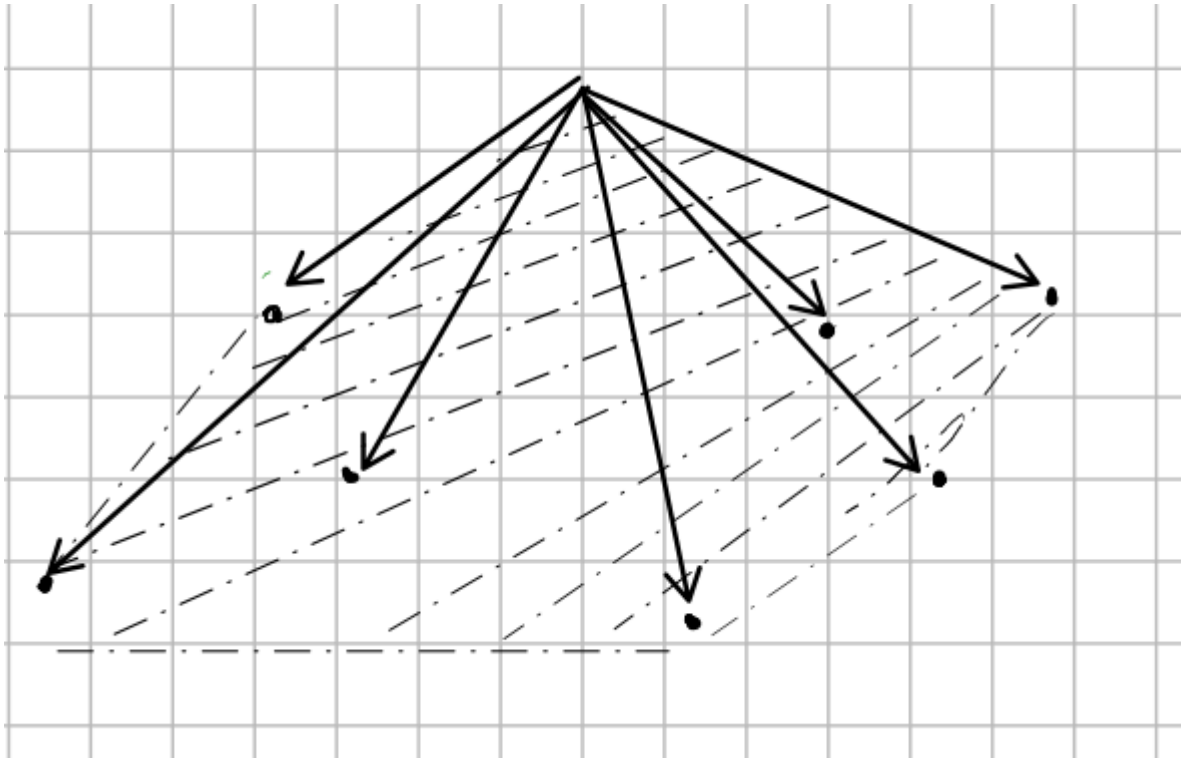
Есть фрилансер (в одном экземпляре), стартапы (до 100 тысяч штук), доход от стартапа в день, опыт от стартапа в день.

Наша цель — заработать хотя бы M денег и хотя бы E опыта.

Над каждым проектом мы можем провести любое количество времени, в том числе нецелочисленное, и получить соответствующую оплату пропорционально времени.

Задача не звучит как геометрическая, однако таковой является. Отметим стартапы как точки на плоскости вида (доход; опыт).

Как измерить профит, если мы поровну поработаем над двумя проектами? Мы получим сумму профитов пополам. Проекты можно брать в любых пропорциях. Можно брать и три, и больше проектов.



Впрочем, больше трёх проектов комбинировать смысла мало.

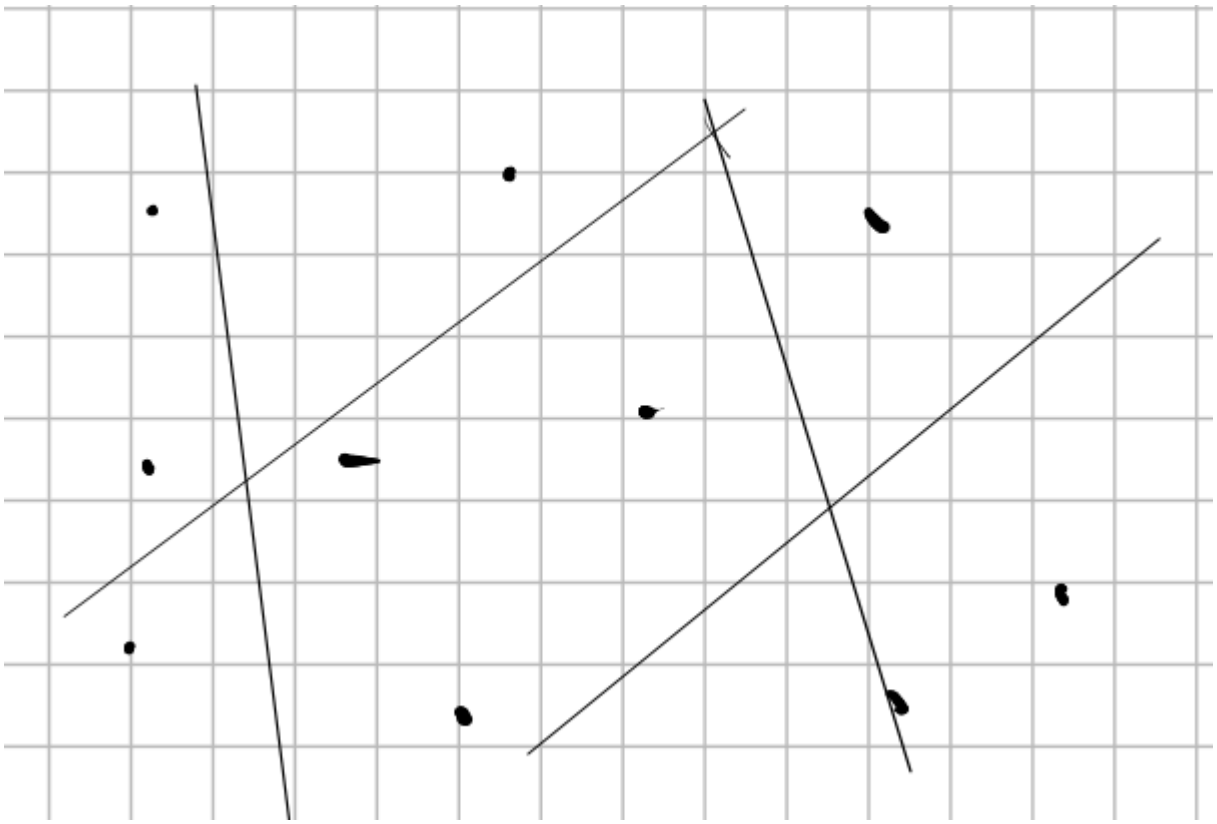
Нам нужно найти точку на оболочке, в которую мы придём, заработав необходимое (или большее) количество профита.

Добавим фиктивные проекты (M, 0) и (0, E). Это решит проблемы “что, если прямая не пересекается с выпуклой оболочек?” и “что, если нам нужно набрать больше денег или опыта?”.

Вариант красивый, но придумать его на контесте сложно. — прим. Вани

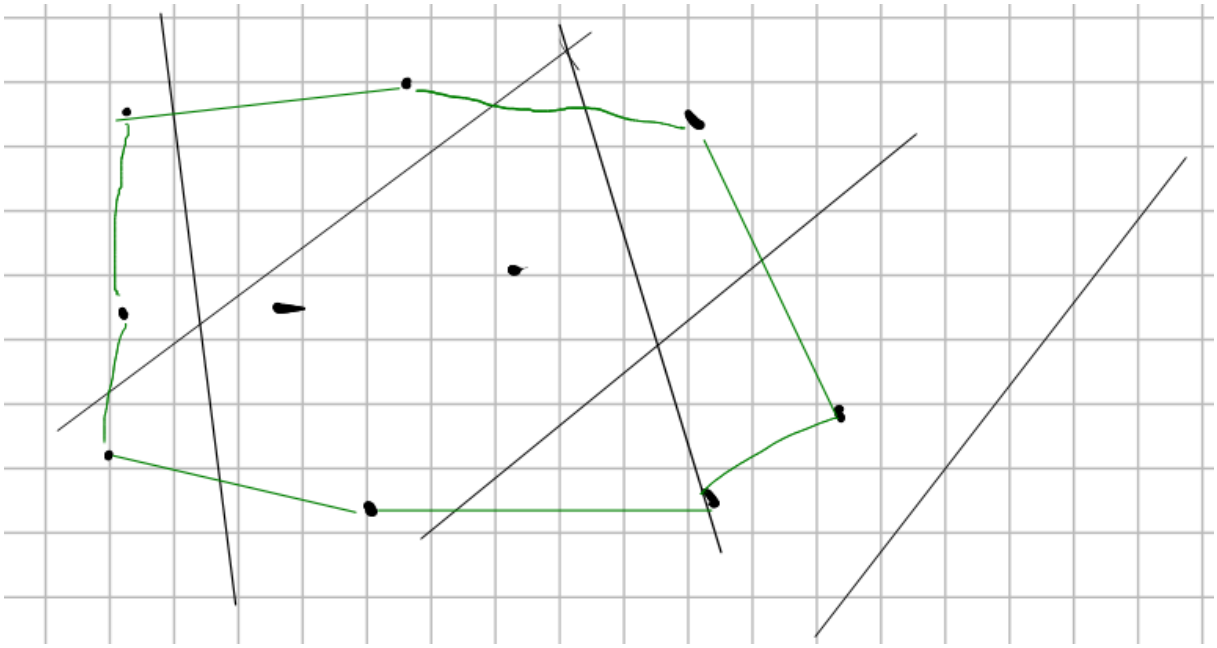
Задача с Кубка России

Есть множество точек на плоскости и набор прямых. Координаты до 10^9. Нужно проверить: верно ли, что все эти точки лежат по одну сторону от этой прямой?



За квадрат мы можем посмотреть относительное расположение точек и прямых.

Но давайте по всем точкам посторим выпуклую оболочку. Её главное свойство таково: если она не пересекается с прямой, то все точки по одну сторону от такой прямой.



А теперь надо понять, пересекается ли прямая с выпуклой оболочкой?

Примечание: данное решение скорее всего не пройдёт с произвольным многоугольником, но в принципе должно работать с произвольным выпуклым многоугольником.

Можно, конечно, перебрать границы выпукой оболочки и прямые и проверить их на пересечение. Это внезапно не квадрат и работает за $O\left(\sqrt{A}\right)$, что следует из размера выпуклой оболочки. Это доказывается следующей вещью:

1. Возьмём самую нижнюю точку выпуклой оболочки
2. Очевидно, что все прочие точки должны быть выше, причём высота очередной точки инкрементируется как минимум на единицу относительно предшествующей.
3. Таким образом, поскольку координаты целочисленные, минимальная высота равна одному и, соответственно, задача работает за корень от высоты минимальной оболочки.

Подставим в уравнение прямой координаты точек. Из предыдущей лекции мы знаем: чем больше число по модулю, тем больше расстояние от точки до прямой:

$$\frac{Ax + By + C}{\sqrt{A^2 + B^2}}$$

Мы можем написать тернарный поиск по выпуклой оболочке.

На будущих занятиях планировалось разобрать продвинутую комбинаторику, но, поскольку не все знакомы с биномиальными коэффициентами, это откладывается