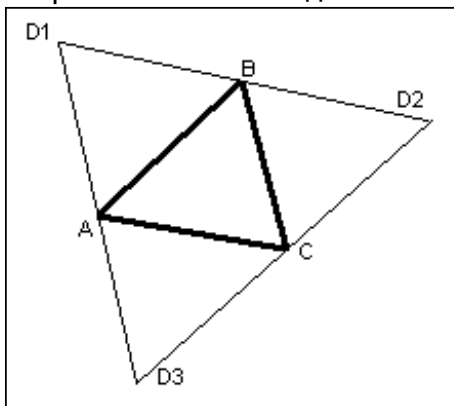


# Параллелограммы

## Решение .

Как обычно, начнём с плохого. Плохое (в смысле малоэффективное, разумеется) решение здесь, естественно, состоит в переборе всевозможных четвёрок точек с проверкой, образуют ли они параллелограмм. Понятно, что сложность алгоритма  $O(N^4)$ , в нашем случае  $N^4 \simeq 4 \cdot 10^{12}$ , так что даже деление на 24 (четыре точки можно переставить 24 способами, а проверять достаточно любую одну перестановку точек) не спасает.

Можно попробовать перебирать тройки точек – ведь три вершины определяют параллелограмм. Впрочем, нет, не совсем определяют – четвёртую вершину можно пристроить тремя различными способами, как показано на рисунке (A, B, C – данные три точки, D1, D2, D3 – три возможности добавить к ним четвёртую вершину параллелограмма):

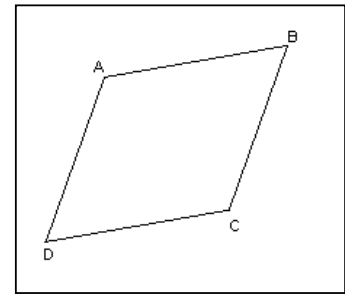


Ладно, три способа – это не много. Итак, выбираем какую-нибудь тройку точек, назовём их A, B и C, вычисляем (понятно, как) координаты точек D1, D2 и D3, и ищем эти D1, D2 и D3 среди данных точек. Каждый удачный поиск даёт нам параллелограмм. Остаются технические детали, вроде того, как не посчитать один и тот же параллелограмм более одного раза, но они легко решаются. Вот только поиск... Но на поиске как раз и можно сэкономить с помощью совершенно стандартного приёма: надо предварительно отсортировать все данные точки и затем каждый раз проверять, имеется ли некоторая точка среди данных с помощью двоичного поиска. Не будем здесь подробно говорить о двоичном поиске, поскольку в данной задаче он не спасает. В самом деле, тройку точек можно выбрать  $O(N^3)$  (точнее  $N \cdot (N-1) \cdot (N-2) / 6 \simeq 5 \cdot 10^8$ ) способами, для каждой тройки двоичный поиск требует  $O(\log N)$  операций, итого, сложность этого алгоритма получается  $O(N^3 \cdot \log N)$ , что тоже неприемлемо много.

Тем не менее, завершая разговор о двоичном поиске, скажем, что он позволяет эффективно решить очень много задач программирования, так что знать его, уметь применять и реализовывать (быстро и корректно!) совершенно необходимо. Так что загляните в *Учебные материалы* – там я выкладываю две главы из уже знакомой вам книги «Жемчужины программирования» Дж.Бентли: в главе 2 приводится довольно много применений двоичного поиска, а в главе 4 строится процедура двоичного поиска (причём не только строится, но и верифицируется; вообще, на примере построения процедуры двоичного поиска демонстрируется сам процесс верификации).

Более эффективное решение даёт нам следующее наблюдение: четырёхугольник ABCD является параллелограммом тогда и только тогда, когда вектора  $\overrightarrow{AB}$  и  $\overrightarrow{DC}$  равны. Иначе говоря, у нас имеется рецепт приготовления (вершин) параллелограмма из четырёх точек, точнее говоря, из двух векторов: два вектора равны – значит они образуют параллелограмм, не равны – образуют не параллелограмм.

Идея есть - надо найти все пары одинаковых векторов, с концами в данных точках. Искать одинаковые элементы – это же “Очень простая задача” ☺ (задача 4-го этапа). И помогает здесь сортировка. Вот и прекрасно – отсортируем все вектора, подсчитаем совпадающие и найдём количество параллелограммов.



Всего имеется  $N \cdot (N-1)$  векторов с вершинами в данных  $N$  точках (замечание:  $N \cdot (N-1)$ , а не  $N \cdot (N-1)/2$ , ведь вектора  $\overrightarrow{AB}$  и  $\overrightarrow{BA}$  - это различные вектора), или  $O(N^2)$  векторов. Сложность сортировки получается  $O(N^2 \cdot \log N^2) = O(N^2 \cdot \log N)$ . Это уже вполне приемлемо. Есть смысл рассмотреть этот алгоритм более тщательно.

Каждая пара одинаковых векторов образует параллелограмм (вот здесь-то и используется то, что никакие три из данных точек не лежат на одной прямой). А вот каждый параллелограмм содержит ровно 4 пары одинаковых векторов: в параллелограмме ABCD это пары  $\overrightarrow{AB}$  и  $\overrightarrow{DC}$ ,  $\overrightarrow{AD}$  и  $\overrightarrow{BC}$ ,  $\overrightarrow{BA}$  и  $\overrightarrow{CD}$ ,  $\overrightarrow{DA}$  и  $\overrightarrow{CB}$ . Вот и славно: подсчитаем количество различных пар одинаковых векторов и разделим это число на 4.

Оценим сложность алгоритма. Сгенерировать массив векторов требует времени  $O(N^2)$ , сортировка имеет сложность  $O(N^2 \cdot \log N)$ , подсчёт пар одинаковых векторов выполняется однократным проходом по (отсортированному!) массиву векторов и, следовательно, тоже требует  $O(N^2)$  времени. Итого, сложность алгоритма равна  $O(N^2 \cdot \log N)$ .

Решение, в принципе, готово, но попробуем его поулучшать. Каждый параллелограмм содержит 4 пары одинаковых векторов. Хорошо бы подсчитывать только одну пару на каждый параллелограмм, или, хотя бы, не все пары совпадающих векторов. Легко видеть, что среди 4 пар одинаковых векторов имеются две пары  $-\overrightarrow{AB} = \overrightarrow{DC}$  и  $\overrightarrow{BA} = \overrightarrow{CD}$ , которые отличаются только направлением, также, как и другие две пары  $\overrightarrow{AD} = \overrightarrow{BC}$  и  $\overrightarrow{DA} = \overrightarrow{CB}$ . Если бы мы для каждой двух точек A и B хранили только один из двух образуемых ими векторов, то от двух пар  $\overrightarrow{AB} = \overrightarrow{DC}$  и  $\overrightarrow{BA} = \overrightarrow{CD}$  осталась бы только одна, также, как и от двух пар  $\overrightarrow{AD} = \overrightarrow{BC}$  и  $\overrightarrow{DA} = \overrightarrow{CB}$ . Но как гарантировать себя от такой неприятности: для точек A и B мы выберем вектор  $\overrightarrow{AB}$ , а для точек C и D – вектор  $\overrightarrow{CD}$ ? Очень просто – будем для любых двух точек выбирать из двух возможных векторов тот, который направлен вправо. Говоря строго, в качестве начальной точки вектора будем выбирать ту, у которой абсцисса меньше. А если абсциссы двух точек совпадают (вектор вертикальный)? Тогда будем брать тот вектор, который направлен вверх. Или, опять-таки, строго говоря, если абсциссы точек совпадают, то в качестве начальной точки будем выбирать точку с меньшей ординатой. А если и ординаты совпадают? ☺☺☺ Дурацкие шуточки. Это противоречит условию – нет трёх точек на прямой, значит и нет двух совпадающих точек ☺

Остаётся только заметить, что в таком случае каждый параллелограмм содержит две пары одинаковых векторов (на приведённом рисунке – это пары  $\overrightarrow{AB} = \overrightarrow{DC}$  и  $\overrightarrow{DA} = \overrightarrow{CB}$ ). Значит, общее количество пар одинаковых векторов надо разделить на два.

В результате количество векторов, которые надо будет сортировать уменьшилось вдвое. А это ускоряет сортировку более, чем вдвое, а третий этап – подсчёт пар одинаковых векторов – вдвое. Пожалуй, на этом и успокоимся...