Лекция 8

Нахождение пары пересекающихся отрезков. Построение выпуклой оболочки

8.1 Нахождение пары пересекающихся отрезков на плоскости

Формулировка задачи. Даны n отрезков на плоскости, заданные координатами начал и концов. Выяснить, пересекаются ли хотя бы какие-то два из них (и найти координаты пересечения).

Предполагаем, что никакие две из данных нам точек не находятся на одной строго вертикальной прямой (от таких точек можно избавиться маленьким поворотом системы координат). Для простоты предполагаем также, что никакие три отрезка не пересекаются в одной точке (от этого тоже можно избавиться «небольшими шевелениями», предварительно слегка удлинив отрезки).

Упражнение 8.1. Доработать алгоритм и доказательство, чтобы избавиться от этих предположений. □

Тривиальный алгоритм перебирал бы $\Omega(n^2)$ пар. Приведенный ниже алгоритм работает быстрее.

Алгоритм. Будем мысленно двигать вертикальную прямую слева направо через всю плоскость и следить за пересечениями отрезков с этой прямой. Конечно, невозможно вычислить пересечения во все моменты времени, но, как будет видно, достаточно следить за пересечениями в те моменты времени, когда прямая проходит через начало или конец какого-либо отрезка; массив этих координат — это просто массив абсцисс данных нам точек.

Мы будем хранить список (номеров) отрезков, пересекаемых нашей вертикальной прямой. Отрезки будут упорядочены по ординате точки пересечения с нашей прямой. Для ускорения модификации этого списка организуем его в виде дерева поиска (B^+- или АВЛ-дерева). Нам будет достаточно следующих операций:

- вставить,
- удалить,
- найти следующий по порядку,
- найти предыдущий по порядку.

Упорядочение отрезков, уже попавших в список, никогда не будет меняться: это может произойти, только если эти два отрезка пересекаются, а эту ситуацию мы вовремя распозна́ем.

Итак, мы просматриваем отсортированный массив абсцисс данных нам точек. На каждом шаге

- если речь идет об ординате *левого* конца какого-либо отрезка, проверяем, пересекается ли он с предыдущим или последующим отрезком из нашего списка, — если нет, добавим его в список;
- если речь идет об ординате *правого* конца какого-либо отрезка, проверим, не пересекаются ли отрезки, которые он разделял, — если нет, удалим его из списка.

(И так, пока не найдем пересекающуюся пару; если ее так и не нашлось, значит ее нет вовсе.)

Время работы алгоритма. Предварительные действия занимают $O(n \log n)$ операций по сортировке. Очевидно, на каждом из $2n$ шагов алгоритм делает константное число операций с деревом поиска (стоимостью $O(\log n)$ операций с числами каждая) и проверяет пересекаемость лишь одной пары отрезков. Итого: $O(n \log n)$ операций.

Замечание 8.1. Ясно, что для того, чтобы найти пересечение одной пары отрезков, достаточно $O(1)$ операций с их координатами. Чтобы оптимизировать эти операции, можно воспользоваться приемами, позволяющими избавиться от операции деления, — такие приемы описаны, например, в книге Кормена, Лейзерсона и Ривеста. Стоит ли их применять, зависит от того, значениями какого типа представлены координаты, а также от архитектуры конкретного компьютера.

Корректность работы алгоритма.

Теорема 8.1. *Приведенный алгоритм корректен.*

Доказательство. Рассмотрим самое левое пересечение. Если в момент, когда мы проходили начало второго отрезка из соответствующей пары, эти отрезки стали соседними (в нашем списке), то мы нашли пересечение. В противном случае между ними были какие-то другие отрезки.

Рассмотрим тогда последний момент перед искомым пересечением, когда мы рассматривали нашу вертикальную прямую. Заметим, что если и в этот момент между нашими отрезками были еще отрезки, то они должны закончиться до пересечения (иначе наше пересечение – не самое левое). Следовательно, в этот момент эти отрезки заканчиваются, а значит, такой отрезок всего один. Согласно соответствующему пункту алгоритма, в этот момент мы найдем искомое пересечение. $\square$

8.2 Построение выпуклой оболочки на плоскости

Выпуклой оболочкой множества точек $\{(x_i, y_i)\}_{i\in I}$ называется множество $\{(\sum_{i\in I}\alpha_i x_i, \sum_{i\in I}\alpha_i y_i) \mid \sum_{i\in I}\alpha_i = 1 \text{ и } \forall i\in I\ \alpha_i \geq 0\}$.

Если I конечно, получится выпуклый многоугольник, вершинами которого являются некоторые из заданных точек (их-то нам и требуется найти), и в котором лежат все заданные точки.

Мысленно привяжем веревку к одной из «внешних» точек и будем обходить вокруг множества наших точек по часовой стрелке — получится как раз искомый многоугольник. Более точно, воспользуемся следующим алгоритмом.

Начинаем с самой нижней точки (самой левой из таковых, если их несколько). Чтобы определить следующую точку, записываем остальные в полярных координатах относительно данной — и выбираем самую левую[1] точку (самую дальнюю из таковых, если их несколько). И так, пока не дойдем до исходной точки.

Пусть дано n точек, а у выпуклой оболочки — h вершин. Указанным образом мы обойдем ровно h вершин, затратив на поиск каждой следующей $O(n)$ операций — итого, $O(nh)$ операций.

[1]Пока идем вверх — это точка, у которой угол $\in [0, \pi)$ и максимален; когда идем вниз — точка, у которой угол $\in [\pi, 2\pi)$ и максимален. Мы «переключаемся» с направления «вверх» на направление «вниз» один раз, когда в интервал $[0, \pi)$ не попадает ни один угол.

Замечание 8.2. Операции над целыми (а следовательно, и рациональными) числами обычно реализуются быстрее и точнее, чем над вещественными (тем более, чем тригонометрические операции). Поэтому вместо углов обычно имеет смысл сравнивать, к примеру, их тангенсы (которые для целочисленных точек являются рациональными числами), не беря арктангенс, и т. д.