ЛКШ, ЛКШ.2018.Август B'
B', конспект лекции

Собрано 21 августа 2018 г. в 21:10

Содержание

Тема #1: Площадь многоугольника

14 августа

1.1. Через трапеции

Научимся находить площадь выпуклого многоугольника.
Выберем порядок обхода. Перебираем вершины в этом порядке. Пусть мы сейчас рассматриваем вершину A, за ней в списке следует B. Если $A.x < B.x$, то прибавим к результату площадь трапеции, с основаниями, пущеными из A и B перпендикулярно к оси абсцисс, иначе — вычтем. Это записывается простой формулой $\frac{(B.x-A.x)\cdot(A.y+B.y)}{2}$
TODO: картика.
Чтобы учесть последнее ребро, запишем самую первую вершину в конец массива.
Результат мог получится как положительный, так и отрицательный. Это зависит от направления обхода. Поэтому не забудем взять по модулю.

```
1 vector<Point> polygon;
2 polygon.push_back(polygon.front());
3 long long S = 0;
4 for (int i = 0; i < n; ++i) {
5     S += (polygon[i+1].x - polygon[i].x) * (polygon[i+1].y + polygon[i].y);
6 }
7 cout << fabs(S / 2.0) << '\n';
```

Этот же способ работает и для невыпуклых прямоугольников.

1.2. Через треугольники

Проведем из точки $(0, 0)$ отрезки ко всем вершинам. Посчитаем ориентированный площади треугольников, образованных ребром и двумя отрезками из начала координат, в порядке обхода. Утверждается, что мы получим то же самое, что и в прошлой задаче (площадь со знаком плюс или минус, в зависимости от порядка обхода).
TODO: картинка

```
1 vector<Point> polygon;
2 polygon.push_back(polygon.front());
3 long long S = 0;
4 for (int i = 0; i < n; ++i) {
5     S += polygon[i] * polygon[i+1]; // векторное произведение
6 }
7 cout << fabs(S / 2.0) << '\n';
```

Оба эти алгоритма работают за $\mathcal{O}(n)$

Тема #2: Принадлежность точки многоугольнику

14 августа

Пустим горизонтальный луч из точки, посчитаем количество пересечений со сторонами многоугольника. Если очо четно, то мы снаружи, а если нечетно, то внутри.
Но тут есть проблема: наш луч может попасть в какую-то вершину, и не понятно, считать это за 2 пересечения или за одно.
TODO: картинка
Эту проблему можно решить. Скажем, что теперь вершины не всегда принадлежат стороне, а именно для стороны AB:

1. $A.y > B.y \Rightarrow A \notin AB$
2. $A.y \leqslant B.y \Rightarrow A \in AB$

Наша задача свелась к n ответам на запрос «Пересекается ли луч с отрезком/полуинтервалом». Допустим, мы умеем проверять пересечение луча и отрезка. Тогда нужно проверить, не по верхней ли координате это пересечение. А это можем сделать одним ифом (сравнить y координаты).
Осталось проверить пересечение луча и отрезка. Мы уже умеем проверять пересечение двух отрезков (см. прошлую лекцию). Скажем, что мы пустили не луч, а очень длинный отрезок (вторая координата $+\infty$).
Если координаты многоугольника целочисленные, то мы смогли решить задачу в целых числах.

Тема #3: Проверка на выпуклость

14 августа

Многоугольник называется выпуклым, если отрезок, соединяющий любые две точки внутри него, целиком лежит внутри.

Равносильное определение: многоугольник выпуклый, если все соседние стороны образуют поворот в одну и ту же сторону.

Второе определение проверять гораздо легче. Направление поворота умеем проверять по знаку векторного произведения.

$\Rightarrow$ знак векторного произведения для всех соседних пар сторон одинаковый $\Leftrightarrow$ многоугольник выпуклый.

Здесь полезно зафиксировать направление обхода, чтобы не мучиться с обработкой случая двух знаков. По часовой стрелке знак минус, против — плюс.

Тема #4: Выпуклая оболочка

14 августа

Выпуклой оболочкой множества точек называется наименьший по площади выпуклый многоугольник, содержащий в себе или на границе все точки этого множества.
Неформальное определение: точки множества — гвоздики. Натягиваем бооольшую резинку вокруг, потом отпускаем.
Можно доказать, что вершины выпуклой оболочки — точки из множества.

4.1. Алгоритм Джарвиса (заворачивания подарка)

На каждом шаге алгоритм будет выбирать очередную точку, лежащую в выпуклой оболочке.
Изначально в выпуклой оболочке будет лежать только нижняя левая точка (она точно есть в выпуклой оболочке и ее можно найти за линию).
Теперь пусть мы взяли i точек. Хотим взять $i+1$-ю. Для этого переберем все невзятые точки. Из них нужно взять ту, угол поворота до которой (в системе координат с началом в последней взятой точке) минимален. Это мы умеем делать векторным произведением.

```
1  vector<Vector> polygon;
2  ...
3  // start - самая левая из самых нижних точек
4  convex_hull.push_back(polygon[start]);
5  polygon.erase(polygon.begin() + start);
6  while (convex_hull.size() == 1 || convex_hull.back() != start) {
7      int idx = 0;
8      for (int i = 1; i < (int) polygon.size(); ++i) {
9          if (Vector(polygon[i] - convex_hull.back()) * Vector(polygon[idx] -
               convex_hull.back()) >= 0) {
10             idx = i;
11         }
12     }
13     convex_hull.push_back(polygon[idx]);
14     polygon.erase(polygon.begin() + idx);
15 }
```

Этот алгоритм работает за $\mathcal{O}(nh)$, где h — размер выпуклой оболочки.
Если мы хотим, чтобы в построенной оболочке не было углов 180 градусов, нужно выбирать из равных по углу самую дальнюю точку

4.2. Алгоритм Грэхема

Давайте снова выберем самую левую из самых нижних точек. Назовем ее `z`. Дальше отсортируем точки по полярному углу, если перенести начало координат в точку `z`.
Как можно это сделать?
Способ первый. Предподсчитать `atan2` для всех точек изначально. Запустить обычный `sort`.
Плюсы: это быстро ($\mathcal{O}(n)$ вызовов `atan2`, быстрая сортировка)
Способ второй. Отсортировать по компаратору (векторное произведение, длина).
Плюсы: целые числа
Минусы: $\mathcal{O}(n \log n)$ раз вызываем не самое быстрое векторное произведение
После того, как мы отсортировали, идем со стеком. Перебираем очередную точку. Хотим взять ее в выпуклую оболочку. Когда мы не можем взять? Когда предпоследняя точка в стеке, послед-

няя точка в стеке и наша точка образуют правый поворот. В таком случае уберем последнюю точку из стека.
TODO: картинка
Чтобы восстановить всю выпуклую оболочку нам удобно положить в конец отсортированного вектора точку `z`.

```
bool is_right_turn(Vector a, Vector b, Vector c) {
    return (b - a) * (c - b) < 0;
}

for (int i = 1; i <= n; ++i) {
    while (st.size() >= 2 && is_right_turn(st[st.size() - 2],
            st.back(), polygon[i]))
        st.pop_back();
    st.push_back(polygon[i]);
}
```

По окончании в стеке останутся только вершины из выпуклой оболочки.
TODO: написать про лямбды.

4.3. Алгоритм Эндрю

Отсортировали точки по x. Выбрали самую левую. Постоили выпуклую оболочку предыдущим алгоритмом для нижнего множества (там должен быть левый поворот) и для верхнего (правый поворот). Объединили две выпуклые оболочки.

Тема #5: Пересечение полуплоскостей

14 августа

TODO