Федеральное государственное автономное образовательное учреждение высшего образования «Казанский (Приволжский) федеральный университет»

Семестровая работа

по алгоритмам и структурам данных

Тема: реализация на языке программирования C++ алгоритма принадлежности точки выпуклому многоугольнику.

Выполнила студентка группы

11-002 Института ИТИС

Хасанова Л.Р.

Казань 2021

В вычислительной геометрии известна задача об определении принадлежности точки выпуклому многоугольнику. На плоскости даны выпуклый многоугольник и точка. Требуется решить вопрос о принадлежности точки данному многоугольнику.

Многоугольник может быть как выпуклым, так и невыпуклым. В последнем случае разные способы определения принадлежности точки многоугольнику могут привести к разным результатам. Различают алгоритмы без предварительной обработки и алгоритмы с предварительной обработкой, в ходе которой создаются некоторые структуры данных, позволяющие в дальнейшем быстрее отвечать на множество запросов о принадлежности точек одному и тому же многоугольнику.

***Основной принцип устройства. Особенности.***

**• Задача:**

Дан выпуклый многоугольник с N вершинами, координаты всех вершин целочисленные; вершины заданы в порядке обхода против часовой стрелки (в противном случае нужно просто отсортировать их). Поступают запросы – точки, и требуется для каждой точки определить, лежит она внутри этого многоугольника или нет (границы многоугольника включаются). Каждый запрос будет выполняться за O(logN). Предварительная обработка многоугольника будет выполняться за O(N).

**• Основной принцип устройства:**

Решение представлено бинарным поиском по углу. Выберем точку с наименьшей координатой «х» (если таких несколько, то выбираем с наименьшим «у»). Относительно этой точки, обозначим её как zero\_value, остальные вершины многоугольника лежат в правой полуплоскости. Заметим, что все вершины многоугольника уже упорядочены по углу относительно точки zero\_value (потому что многоугольник выпуклый, и уже упорядочен против часовой стрелки), причём все углы находятся в промежутке (-π/2; π/2].

Пусть на вход поступает очередной запрос – некоторая точка «Р». Рассмотрим её полярный угол относительно точки zero\_value. Найдём бинарным поиском две такие соседние вершины «L» и «R» многоугольника, что полярный угол «P» лежит между полярными углами «L» и «R». Таким образом находится тот сектор многоугольника, в котором лежит точка «P», и остаётся только проверить, лежит ли точка «P» в треугольнике (zero\_value, L, R). Это можно сделать, например, с помощью ориентированной площади треугольника и предиката "по часовой стрелке".

**• Пояснение к ориентированной площади треугольника:**

Пусть даны три точки: р1, р2, р3. Найдём значение знаковой площади S треугольника р1р2р3, т.е. площади этого треугольника, взятой со знаком плюс или минус в зависимости от типа поворота, образуемого точками р1, р2, р3: против часовой стрелки или по ней соответственно. Если уметь вычислять такую знаковую ("ориентированную") площадь, то можно находить обычную площадь любого треугольника, а также сможем проверять, по часовой стрелке или против направлена какая-либо тройка точек.

Возвращаясь к задаче, понятно, что достаточно посмотреть, по часовой стрелке или против находится тройка вершин (R, L, P). Таким образом, за O(logN) находится сектор многоугольника, а затем за O(1) проверяется принадлежность точки треугольнику, и, следовательно, требуемая асимптотика достигнута. Предварительная обработка многоугольника заключается только в том, чтобы посчитать полярные углы для всех точек, хотя, эти вычисления тоже можно перенести на этап бинарного поиска.

**• Особенности алгоритма:**

Учитывая, что изначально все координаты являются целочисленными, можно получить решение, не использующее дробной арифметики. Заметим, что полярный угол точки (X, Y) относительно начала координат однозначно определяется дробью Y/X, при условии, что точка находится в правой полуплоскости. Более того, если у одной точки полярный угол меньше, чем у другой, то и дробь Y1/X1 будет меньше Y2/X2, и наоборот. Так, для сравнения полярных углов двух точек достаточно сравнить дроби Y1/X1 и Y2/X2, что уже можно выполнить в целочисленной арифметике.

***Оценка временной сложности.***

**• Теорема:** проверка принадлежности точки выпуклому n-угольнику может быть выполнена за время O(logN) при затратах памяти O(n).

**• Доказательство:** Принадлежность точки выпуклому или n -угольнику может быть определена при помощи двоичного поиска. Двоичный (бинарный) поиск (также известен как метод деления пополам) — классический алгоритм поиска элемента в отсортированном массиве (векторе), использующий дробление массива на половины. Следовательно, из-за двоичного поиска проверка принадлежности точки выпуклому n-угольнику выполняется за время O(logN).

*\* Примечание:* в случае простого n-угольника принадлежности точки данному n-угольнику выполняется за время O(N).

***Выводы. Плюсы и минусы алгоритма, его применимость.***

**Выводы:**

Задача об определении принадлежности точки выпуклому многоугольнику имеет свои плюсы, но также присутствуют и минусы. Эта задача применима в различных областях, поэтому существует большое количество алгоритмов для ее решения. Также в общем случае, помимо выпуклых многоугольников, имеют применение задачи об определении принадлежности точки произвольному n-угольнику, многограннику и другим геометрическим фигурам.

**• Плюсы:**

1) существуют разные способы определения принадлежности точки многоугольнику (метод трассировки луча, метод суммировано углов, тригонометрический алгоритм и другие).

2) в отличие от простого n-угольника, в алгоритме принадлежность точки выпуклому n-угольнику не возникает затруднение в вырожденном случае, когда луч пересекает вершину многоугольника.

3) в алгоритмах с предварительной обработкой создаются некоторые структуры данных, позволяющие в дальнейшем быстрее отвечать на множество запросов о принадлежности разных точек одному и тому же многоугольнику.

4) в отличие от простого n-угольника, принадлежность точки выпуклому n-угольнику определяется за время O(logN).

**• Минусы:**

1) в случае невыпуклого n-угольника разные способы определения принадлежности точки многоугольнику могут привести к разным результатам.

2) из-за того, что разные способы определения принадлежности точки многоугольнику могут привести к разным результатам, приходится предварительно обрабатывать некоторые алгоритмы (это так называемые алгоритмы с предварительной обработкой)

3) предварительная обработка требует O(*N*) времени.

**• Применимость:**

Проверка принадлежности точки многоугольнику или многограннику необходима во многих алгоритмах, например, в булевых операциях над многоугольниками или многогранниками. А проверка принадлежности точки многограннику используется, например, для описания слоев в геоинформационных системах.

***Список использованной литературы.***

1) Препарата, Шеймос. Вычислительная геометрия: введение.

2) <https://kpfu.ru/staff_files/F_246194882/Lection1.pdf>

3) <https://ru.wikipedia.org/wiki/Задача_о_принадлежности_точки_многоугольнику>

4) Джалиашвили З. О., Подольский А. А. Алгоритмы анализа ответов в тестовых заданиях с

графическим интерфейсом.

5) <https://cyberleninka.ru/article/n/algoritm-opredeleniya-prinadlezhnosti-tochki-mnogougolniku-obschego-vida-ili-mnogogranniku-s-treugolnymi-granyami/viewer>

6) Ласло М. Вычислительная геометрия и компьютерная графика на C++.

7) <https://dic.academic.ru/dic.nsf/ruwiki/209337>

***Код программы.***

#include <iostream>

#include <vector>

#include <algorithm>

#include <stdio.h>

#include <chrono>

using namespace std;

using namespace std::chrono;

bool in = false;

struct point {

long long x, y;

};

struct angle {

long long a, b;

};

bool operator < (const angle& p, const angle& query) {

if (p.b == 0 && query.b == 0)

return p.a < query.a;

return p.a \* 1ll \* query.b < p.b \* 1ll \* query.a;

}

long long square(point& a, point& b, point& c) {

return a.x \* 1ll \* (b.y - c.y) + b.x \* 1ll \* (c.y - a.y) + c.x \* 1ll \* (a.y - b.y);

}

void tests() {

long n;

cout << "Enter n: ";

cin >> n;

vector<point> p(n);

long zero\_id = 0;

for (int j = 0; j < n; ++j) {

scanf\_s("%d%d", &p[j].x, &p[j].y);

if (p[j].x < p[zero\_id].x || p[j].x == p[zero\_id].x && p[j].y < p[zero\_id].y) {

zero\_id = j;

}

}

point zero\_value = p[zero\_id];

rotate(p.begin(), p.begin() + zero\_id, p.end());

p.erase(p.begin());

--n;

vector<angle> a(n);

for (int k = 0; k < n; ++k) {

a[k].a = p[k].y - zero\_value.y;

a[k].b = p[k].x - zero\_value.x;

if (a[k].a == 0) {

a[k].b = a[k].b < 0 ? -1 : 1;

}

}

point query;

cout << "Enter the coordinates of the point: ";

cin >> query.x >> query.y;

in = false;

if (query.x >= zero\_value.x) {

if (query.x == zero\_value.x && query.y == zero\_value.y) {

in = true;

}

else {

angle my = { query.y - zero\_value.y, query.x - zero\_value.x };

if (my.a == 0) {

my.b = my.b < 0 ? -1 : 1;

}

vector<angle>::iterator it = upper\_bound(a.begin(), a.end(), my);

if (it == a.end() && my.a == a[n - 1].a && my.b == a[n - 1].b) {

it = a.end() - 1;

}

if (it != a.end() && it != a.begin()) {

long long p1 = int(it - a.begin());

if (square(p[p1], p[p1 - 1], query) <= 0) {

in = true;

}

}

}

}

}

int main() {

for (int i = 1; i <= 50; i++) {

cout << "Test #" << i << endl;

auto startMain = high\_resolution\_clock::now();

tests();

auto stopMain = high\_resolution\_clock::now();

if (in) {

cout << "YES";

}

else {

cout << "NO";

}

cout << "" << endl;

cout << "Time taken by algorithm: "

<< duration\_cast<milliseconds>(stopMain - startMain).count() << endl << endl;

}

}

***Оценка сложности алгоритма.***



 