ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ

“КАЗАНСКИЙ (ПРИВОЛЖСКИЙ) ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ”

ИНСТИТУТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ И ИНТЕЛЕКТУАЛЬНЫХ СИСТЕМ

ОТЧЁТ ПО АЛГОРИТМУ

***“Проверка принадлежности точки выпуклому многоугольнику за логарифмическое время”***

Выполнила: Юмадилова Алсу, гр. 11-002

г.Казань

2021

**1) Название алгоритма:** Проверка принадлежности точки выпуклому многоугольнику за логарифмическое время

**2) Основной принцип устройства. Особенности**

Дан выпуклый многоугольник с N вершинами, координаты всех вершин целочисленны, 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).

**Алгоритм:** Задача решается по бинарному поиску по углу.

Выберем точку с наименьшей координатой X (если таких несколько, то выбираем самую нижнюю, т.е. с наименьшим Y). Относительно этой точки, обозначим её Zero, все остальные вершины многоугольника лежат в правой полуплоскости. Далее, заметим, что все вершины многоугольника уже упорядочены по углу относительно точки Zero (это вытекает из того, что многоугольник выпуклый, и уже упорядочен против часовой стрелки), причём все углы находятся в промежутке (-π/2 ; π/2].

Пусть поступает очередной запрос - некоторая точка P. Рассмотрим её полярный угол относительно точки Zero. Найдём бинарным поиском две такие соседние вершины L и R многоугольника, что полярный угол P лежит между полярными углами L и R. Тем самым мы нашли тот сектор многоугольника, в котором лежит точка P, и нам остаётся только проверить, лежит ли точка P в треугольнике (Zero,L,R). Это можно сделать, например, с помощью [Ориентированной площади треугольника и Предиката "По часовой стрелке"](https://e-maxx.ru/algo/oriented_area), достаточно посмотреть, по часовой стрелке или против находится тройка вершин (R,L,P).

**Особенности:** чтобы определять полярный угол, можно воспользоваться стандартной функцией atan2. Тем самым мы получим очень короткое и простое решение, однако взамен могут возникнуть проблемы с точностью.

Учитывая, что изначально все координаты являются целочисленными, можно получить решение, вообще 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 обратно.

Таким образом, для сравнения полярных углов двух точек нам достаточно сравнить дроби Y1/X1 и Y2/X2, что уже можно выполнить в целочисленной арифметике.

**3) Оценка временной сложности**

Время ответа на запрос о принадлежности точки выпуклому *N*-угольнику равно О(log *N*) при затрате O(*N*) памяти и O(*N*) времени на предварительную обработку.

Доказательство: так как мы рассматриваем углы, то наш многоугольник оказывается разбитым на N клиньев. Каждый клин разбит на 2 части одним из ребер Р. Одна из этих частей лежит целиком внутри Р, другая—целиком снаружи. Считая *q* началом полярных координат, мы можем отыскать тот клин, где лежит точка z*,* проведя один раз двоичный поиск, поскольку лучи следуют в порядке возрастания их углов. После нахождения клина остается только сравнить z с тем единственным ребром из Р, которое разрезает этот клин, и решить, лежит ли z внутри *Р.*

Таким образом, мы за O (log N) находим сектор многоугольника, а затем за O (1) проверяем принадлежность точки треугольнику, и, следовательно, требуемая асимптотика достигнута. Предварительная обработка многоугольника заключается только в том, чтобы предпочитать полярные углы для всех точек.

**4) Выводы. Плюсы и минусы алгоритма. Применимость.**

Алгоритм имеет широкое распространение. Помимо выпуклых многоугольников, имеют применение задачи об определении принадлежности точки произвольному n-угольнику, многограннику и другим геометрическим фигурам.

Плюсы:

* Время работы O(logN)
* 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.
* Алгоритм можно реализовать разными способами (метод трассировки луча, метод суммирования углов, тригонометрический алгоритм)

Минусы:

* Если многоугольник невыпуклый, разные способы определения принадлежности точки многоугольнику могут привести к разным результатам.
* 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.

5) Список литературы:

<https://ru.wikibooks.org/wiki/Реализации_алгоритмов/Задача_о_принадлежности_точки_многоугольнику>

<https://ru.wikipedia.org/wiki/Задача_о_принадлежности_точки_многоугольнику#:~:text=В%20вычислительной%20геометрии%20известна%20задача,как%20выпуклым%2C%20так%20и%20невыпуклым>

<https://math-lessons.ru/computers_graphics/igs/63.htm>

<https://pastebin.com/BD0Yvv28>

<https://habr.com/ru/post/144571/>

6) Код программы

#include <iostream>  
#include <vector>  
  
using namespace std;  
  
struct point {  
 int x, y;  
};  
  
struct an {  
 int a, b;  
};  
  
bool operator < (const an & p, const an & q) {  
 if (p.b == 0 && q.b == 0)  
 return p.a < q.a;  
 return p.a \* 1ll \* q.b < p.b \* 1ll \* q.a;  
}  
  
long long sqrye (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);  
}  
  
int main() {  
 int n; // количество вершин многоугольника  
 cin >> n;  
 vector<point> p(n);  
 int zero\_id = 0; // переменная хранит номер самой низкой точки на координатной плоскости  
 // ввод очередной точки многоугольника и нахождение самой низкой  
 for (int i = 0; i < n; ++i) {  
 cin >> p[i].x;  
 cin >> p[i].y;  
 if (p[i].x < p[zero\_id].x || p[i].x == p[zero\_id].x && p[i].y < p[zero\_id].y)  
 zero\_id = i;  
 }  
  
 point zero = p[zero\_id];  
  
 rotate(p.begin(), p.begin() + zero\_id, p.end());  
 p.erase(p.begin());  
 --n;  
  
 // вычисляем угол  
 vector<an> a(n);  
 for (int i = 0; i < n; ++i) {  
 a[i].a = p[i].y - zero.y;  
 a[i].b = p[i].x - zero.x;  
 if (a[i].a == 0)  
 a[i].b = a[i].b < 0 ? -1 : 1;  
 }  
  
 point t{};  
  
 // ввод координат точки  
 cin >> t.x;  
 cin >> t.y;  
 bool flag = false;  
 if (t.x >= zero.x)  
 if (t.x == zero.x && t.y == zero.y)  
 flag = true;  
 else {  
 an an = {t.y - zero.y, t.x - zero.x };  
 if (an.a == 0)  
 an.b = an.b < 0 ? -1 : 1;  
 auto it = upper\_bound (a.begin(), a.end(), an);  
 if (it == a.end() && an.a == a[n - 1].a && an.b == a[n - 1].b)  
 it = a.end()-1;  
 if (it != a.end() && it != a.begin()) {  
 int p1 = int (it - a.begin());  
 if (sqrye(p[p1], p[p1 - 1], t) <= 0)  
 flag = true;  
 }  
 }  
  
 if(flag) {  
 cout << "The point " << t.x << " " << t.y << " belongs the polygon";  
 } else {  
 cout << "The point " << t.x << " " << t.y << " not belongs the polygon";  
 }  
}

**7)**

Входные данные: <https://github.com/Alsu11/ALSUAiSD/tree/main/Semestrovaya2/Tests>

**8)**

График зависимости времени от количества вершин многоугольника



|  |  |
| --- | --- |
| Количство вершин мн-ка | Время работы |
| 17 | 2 |
| 18 | 3 |
| 25 | 5 |
| 32 | 5 |
| 42 | 6 |
| 51 | 5 |
| 61 | 6 |
| 69 | 7 |
| 78 | 6 |
| 84 | 7 |
| 90 | 7 |
| 98 | 6 |
| 102 | 7 |
| 112 | 8 |
| 123 | 7 |
| 134 | 7 |
| 145 | 8 |
| 156 | 8 |
| 162 | 8 |
| 174 | 7 |
| 186 | 7 |
| 193 | 8 |
| 204 | 8 |
| 213 | 8 |
| 226 | 7 |
| 234 | 8 |
| 243 | 6 |
| 257 | 8 |
| 264 | 8 |
| 276 | 7 |
| 286 | 7 |
| 293 | 9 |
| 304 | 9 |
| 312 | 8 |
| 326 | 8 |
| 331 | 7 |
| 345 | 8 |
| 356 | 8 |
| 367 | 6 |
| 371 | 7 |
| 385 | 8 |
| 393 | 8 |
| 403 | 8 |
| 413 | 9 |
| 427 | 9 |
| 435 | 6 |
| 445 | 7 |
| 456 | 8 |
| 467 | 6 |
| 472 | 7 |
| 485 | 7 |
| 493 | 8 |
| 502 | 8 |
| 512 | 8 |
| 522 | 9 |
| 538 | 9 |
| 551 | 9 |
| 569 | 8 |
| 578 | 9 |
| 583 | 9 |