Last update: August 27, 2023 Translated From: e-maxx.ru

Check if point belongs to the convex polygon in $O(\log N)$

Consider the following problem: you are given a convex polygon with integer vertices and a lot of queries. Each query is a point, for which we should determine whether it lies inside or on the boundary of the polygon or not. Suppose the polygon is ordered counter-clockwise. We will answer each query in $O(\log n)$ online.

Algorithm

Let's pick the point with the smallest x-coordinate. If there are several of them, we pick the one with the smallest y-coordinate. Let's denote it as p_0. Now all other points $p_1, \dots, p_n$ of the polygon are ordered by their polar angle from the chosen point (because the polygon is ordered counter-clockwise).

If the point belongs to the polygon, it belongs to some triangle p_0, p_i, p_{i+1} (maybe more than one if it lies on the boundary of triangles). Consider the triangle p_0, p_i, p_{i+1} such that p belongs to this triangle and i is maximum among all such triangles.

There is one special case. p lies on the segment (p_0, p_n). This case we will check separately. Otherwise all points p_j with $j \le i$ are counter-clockwise from p with respect to p_0, and all other points are not counter-clockwise from p. This means that we can apply binary search for the point p_i, such that p_i is not counter-clockwise from p with respect to p_0, and i is maximum among all such points. And afterwards we check if the points is actually in the determined triangle.

The sign of $(a - c) \times (b - c)$ will tell us, if the point a is clockwise or counter-clockwise from the point b with respect to the point c. If $(a - c) \times (b - c) > 0$, then the point a is to the right of the vector going from c to b, which means clockwise from b with respect to c. And if $(a - c) \times (b - c) < 0$, then the point is to the left, or counter clockwise. And it is exactly on the line between the points b and c.

Back to the algorithm: Consider a query point p. Firstly, we must check if the point lies between p_1 and p_n. Otherwise we already know that it cannot be part of the polygon. This can be done by checking if the cross product $(p_1 - p_0) \times (p - p_0)$ is zero or has the same sign with $(p_1 - p_0) \times (p_n - p_0)$, and $(p_n - p_0) \times (p - p_0)$ is zero or has the same sign with $(p_n - p_0) \times (p_1 - p_0)$. Then we handle the special case in which p is part of the line (p_0, p_1). And then we can binary search the last point from $p_1, \dots p_n$ which is not counter-clockwise from p with respect to p_0. For a single point p_i this condition can be checked by checking that $(p_i - p_0) \times (p - p_0) \le 0$. After we found such a point p_i, we must test if p lies inside the triangle p_0, p_i, p_{i+1}. To test if it belongs to the triangle, we may simply check that $|(p_i - p_0) \times (p_{i+1} - p_0)| = |(p_0 - p) \times (p_i - p)| + |(p_i - p) \times (p_{i+1} - p)| + |(p_{i+1} - p) \times (p_0 - p)|$. This checks if the area of the triangle p_0, p_i, p_{i+1} has to exact same size as the sum of the sizes of the triangle p_0, p_i, p, the triangle p_0, p, p_{i+1} and the triangle p_i, p_{i+1}, p. If p is outside, then the sum of those three triangle will be bigger than the size of the triangle. If it is inside, then it will be equal.

Implementation

The function `prepare` will make sure that the lexicographical smallest point (smallest x value, and in ties smallest y value) will be p_0, and computes the vectors $p_i - p_0$. Afterwards the function `pointInConvexPolygon` computes the result of a query. We additionally remember the point p_0 and translate all queried points with it in order compute the correct distance, as vectors don't have an initial point. By translating the query points we can assume that all vectors start at the origin $(0, 0)$, and simplify the computations for distances and lengths.

```
struct pt {
    long long x, y;
    pt() {}
    pt(long long _x, long long _y) : x(_x), y(_y) {}
    pt operator+(const pt &p) const { return pt(x + p.x, y + p.y); }
    pt operator-(const pt &p) const { return pt(x - p.x, y - p.y); }
    long long cross(const pt &p) const { return x * p.y - y * p.x; }
    long long dot(const pt &p) const { return x * p.x + y * p.y; }
    long long cross(const pt &a, const pt &b) const { return (a - *this).cross(b - *this); }
    long long dot(const pt &a, const pt &b) const { return (a - *this).dot(b - *this); }
    long long sqrLen() const { return this->dot(*this); }
};

bool lexComp(const pt &l, const pt &r) {
    return l.x < r.x || (l.x == r.x && l.y < r.y);
}

int sgn(long long val) { return val > 0 ? 1 : (val == 0 ? 0 : -1); }

vector<pt> seq;
pt translation;
int n;

bool pointInTriangle(pt a, pt b, pt c, pt point) {
```

```cpp
    long long s1 = abs(a.cross(b, c));
    long long s2 = abs(point.cross(a, b)) + abs(point.cross(b, c)) + abs(point.cross(c, a));
    return s1 == s2;
}

void prepare(vector<pt> &points) {
    n = points.size();
    int pos = 0;
    for (int i = 1; i < n; i++) {
        if (lexComp(points[i], points[pos]))
            pos = i;
    }
    rotate(points.begin(), points.begin() + pos, points.end());

    n--;
    seq.resize(n);
    for (int i = 0; i < n; i++)
        seq[i] = points[i + 1] - points[0];
    translation = points[0];
}

bool pointInConvexPolygon(pt point) {
    point = point - translation;
    if (seq[0].cross(point) != 0 &&
            sgn(seq[0].cross(point)) != sgn(seq[0].cross(seq[n - 1])))
        return false;
    if (seq[n - 1].cross(point) != 0 &&
            sgn(seq[n - 1].cross(point)) != sgn(seq[n - 1].cross(seq[0])))
        return false;

    if (seq[0].cross(point) == 0)
        return seq[0].sqrLen() >= point.sqrLen();

    int l = 0, r = n - 1;
    while (r - l > 1) {
        int mid = (l + r) / 2;
        int pos = mid;
        if (seq[pos].cross(point) >= 0)
            l = mid;
        else
            r = mid;
    }
    int pos = l;
    return pointInTriangle(seq[pos], seq[pos + 1], pt(0, 0), point);
}
```

Problems

SGU253 Theodore Roosevelt Codeforces 55E Very simple problem

Contributors:

SYury (81.45%) jakobkogler (11.29%) adamant-pwn (4.84%) dallasyan (1.61%) fcnoronha (0.81%)