USACO JAN14 Problem 'curling' Analysis

by Nathan Pinsker

We will only consider the case of finding how many blue points are contained in triangles formed by red points, as the reverse case is effectively identical.

Given N red points on the plane, there are an overwhelming number of triangles to consider, not to mention we must check all of our N blue points. However, we can reduce this by noting that any blue stone that counts as a point for red must at least be in the convex hull formed by the N red points. With a little bit of thought, one can see that the total area covered by all possible triangles is *exactly* that of the convex hull formed by our N red points. Thus, we can start by using any of the standard algorithms to find the convex hull of our red points.

Given this convex hull, we have now reduced our problem to checking whether each of our N blue points is inside it. This can be done by storing the hull either in cyclic or left to right order.

To consider the hull in cyclic order, we designate an arbitrary point O inside the convex hull, then translate every point on the plane so that O is at the origin. By doing this, we are guaranteed that our convex hull contains the origin, meaning that the points on our convex hull can be easily arranged in order of increasing angle with the x-axis. Thus, given any blue point, we can just find its angle with the x-axis, and easily find the "slice" of the convex hull that it would have to be in, if it is inside the hull at all.

The left-to-right order breaks the hull into trapezoids based on the x values of all the vertices. Each trapezoid covers a unique range of x values, so to locate a point we simply search for the x range that contains its x value.

In either case, We can check in O(logn) time whether our point is in that slice by using some simple computational geometry. This gives us an overall runtime of O(N log N), as each query takes O(log N) time and there are exactly N of them.

Here is Bruce Merry's code:

```
#include <iostream>
#include <fstream>
#include <vector>
#include <complex>
#include <algorithm>

using namespace std;

typedef long long ll;
typedef complex<long long> pnt;

template<typename T>
int size(const T &t)
{
    return t.size();
}

static inline ll dot(const pnt &a, const pnt &b)
{
    return real(conj(a) * b);
}
```

```
static inline ll cross(const pnt &a, const pnt &b)
{
    return imag(conj(a) * b);
}

static inline ll cross(const pnt &a, const pnt &b, const pnt &c)
{
    return cross(b - a, c - a);
}

struct compare_xy
{
    bool operator()(const pnt &a, const pnt &b) const
    {
        if (a.imag() != b.imag())
            return a.imag() < b.imag();
        else
            return a.real() < b.real();
    }
};

struct compare_angle
{
    pnt base;

    compare_angle(const pnt &base) : base(base) {}

    bool operator()(const pnt &a, const pnt &b) const
    {
        ll c1 = cross(base, a, b);
        if (c1 != 0)
            return c1 > 0;
        else
            return dot(a - base, a - base) < dot(b - base, b - base);
    }
};

static vector<pnt> chull(vector<pnt> &in)
{
    swap(in[0], *min_element(in.begin(), in.end(), compare_xy()));
    sort(in.begin() + 1, in.end(), compare_angle(in[0]));

    vector<pnt> h;
    h.push_back(in[0]);
    for (size_t i = 1; i < in.size(); i++)
    {
        const pnt &p = in[i];
        while (h.size() >= 2 && cross(h[h.size() - 2], h[h.size() - 1], p) <=
0)
            h.pop_back();
        h.push_back(p);
    }
    return h;
}

static inline int wrap(int x, int H)
{
    return x >= H ? x - H : x;
}

static int score(const vector<pnt> &h, vector<pnt> &s)
{
```

```
    int H = size(h);
    int bl = 0;
    int tr = max_element(h.begin(), h.end(), compare_xy()) - h.begin();
    int br = bl;
    int tl = tr;
    if (h[bl].imag() == h[tr].imag())
    {
        br = tr;
        tl = bl;
    }
    else
    {
        while (h[br + 1].imag() == h[bl].imag())
            br++;
        while (h[wrap(tl + 1, H)].imag() == h[tr].imag())
            tl = wrap(tl + 1, H);
    }

    sort(s.begin(), s.end(), compare_xy());
    int r = 0;
    int l = h.size() - 1;
    int total = 0;
    for (size_t i = 0; i < s.size(); i++)
    {
        const pnt &p = s[i];
        bool hit = false;
        if (p.imag() >= h[0].imag() && p.imag() <= h[tr].imag())
        {
            if (p.imag() == h[bl].imag())
                hit = p.real() >= h[bl].real() && p.real() <= h[br].real();
            else if (p.imag() == h[tr].imag())
                hit = p.real() >= h[tl].real() && p.real() <= h[tr].real();
            else
            {
                while (p.imag() >= h[r + 1].imag())
                    r++;
                while (p.imag() >= h[l].imag())
                    l--;
                hit = (cross(h[r], h[wrap(r + 1, H)], p) >= 0
                    && cross(h[l], h[wrap(l + 1, H)], p) >= 0);
            }
        }
        total += hit;
    }
    return total;
}

static int solve(vector<pnt> &A, vector<pnt> &B)
{
    const vector<pnt> hull = chull(A);
    return score(hull, B);
}

int main()
{
    ifstream in("curling.in");
    ofstream out("curling.out");
    int N;
    in >> N;
    vector<pnt> A(N), B(N);
    for (int i = 0; i < N; i++)
    {
        int x, y;
```

```
        in >> x >> y;
        A[i] = pnt(x, y);
    }
    for (int i = 0; i < N; i++)
    {
        int x, y;
        in >> x >> y;
        B[i] = pnt(x, y);
    }

    int sA = solve(A, B);
    int sB = solve(B, A);
    out << sA << ' ' << sB << '\n';
    return 0;
}
```