

ICPC NTHU valderyayaorz

llshang valderyaya Penguin07

September 14, 2022

Contents

1 Geometry

1.1 Point.h	1
1.2 ConvexHull.h	2

1 Geometry

1.1 Point.h

```
1 template<class T>
2 class Point {
3 public:
4     T x, y;
5     Point() : x(0), y(0) {}
6     Point(const T& a, const T& b) : x(a), y(b) {}
7     template<class U>
8     explicit Point(const Point<U>& p) :
9     ↪ x(static_cast<T>(p.x)), y(static_cast<T>(p.y))
10    ↪ {}
11    Point(const pair<T, T>& p) : x(p.first),
12    ↪ y(p.second) {}
13    Point(const complex<T>& p) : x(real(p)),
14    ↪ y(imag(p)) {}
15    explicit operator pair<T, T>() const {
16        return pair<T, T>(x, y);
17    }
18    explicit operator complex<T>() const {
19        return complex<T>(x, y);
20    }
21    inline Point& operator+=(const Point& rhs) {
22        x += rhs.x;
23        y += rhs.y;
24        return *this;
25    }
26    inline Point& operator-=(const Point& rhs) {
27        x -= rhs.x;
28        y -= rhs.y;
29        return *this;
30    }
31    inline Point& operator*=(const T& rhs) {
32        x *= rhs;
33        y *= rhs;
34        return *this;
35    }
36    inline Point& operator/=(const T& rhs) {
37        x /= rhs;
38        y /= rhs;
39        return *this;
40    }
41    template<class U>
42    inline Point& operator+=(const Point<U>& rhs) {
```

```
39         return *this += Point<T>(rhs);
40     }
41     template<class U>
42     inline Point& operator-=(const Point<U>& rhs) {
43         return *this -= Point<T>(rhs);
44     }
45     inline Point operator+() const {
46         return *this;
47     }
48     inline Point operator-() const {
49         return Point(-x, -y);
50     }
51     inline Point operator+(const Point& rhs) {
52         return Point(*this) += rhs;
53     }
54     inline Point operator-(const Point& rhs) {
55         return Point(*this) -= rhs;
56     }
57     inline Point operator*(const T& rhs) {
58         return Point(*this) *= rhs;
59     }
60     inline Point operator/(const T& rhs) {
61         return Point(*this) /= rhs;
62     }
63     inline bool operator==(const Point& rhs) {
64         return x == rhs.x && y == rhs.y;
65     }
66     inline bool operator!=(const Point& rhs) {
67         return !(*this == rhs);
68     }
69     inline T dist2() const {
70         return x * x + y * y;
71     }
72     inline long double dist() const {
73         return sqrt(dist2());
74     }
75     inline Point unit() const {
76         return *this / this->dist();
77     }
78     inline long double angle() const {
79         return atan2(y, x);
80     }
81     inline friend T dot(const Point& lhs, const
82     ↪ Point& rhs) {
83         return lhs.x * rhs.x + lhs.y * rhs.y;
84     }
85     inline friend T cross(const Point& lhs, const
86     ↪ Point& rhs) {
87         return lhs.x * rhs.y - lhs.y * rhs.x;
88     }
89     inline friend Point dot_cross(const Point& lhs,
90     ↪ const Point& rhs) {
91         return Point(dot(lhs, rhs), cross(lhs,
92         ↪ rhs));
93     }
94 }
```

```

90 };
91 template<class T>
92 istream& operator>>(istream& in, Point<T>& p) {
93     return in >> p.x >> p.y;
94 }

```

1.2 ConvexHull.h

```

1 // @return the points of the convex hull in
   ↪ clock-wise order
2 template<class T>
3 vector<Point<T>> ConvexHull(vector<Point<T>>
   ↪ points) {
4     const int n = (int) points.size();
5     sort(points.begin(), points.end(), [](const
   ↪ Point<T>& a, const Point<T>& b) {
6         if(a.x == b.x) {
7             return a.y < b.y;
8         }
9         return a.x < b.x;
10    });
11    auto build = [&]() {
12        vector<Point<T>> upper;
13        upper.push_back(points[0]);
14        upper.push_back(points[1]);
15        for(int i = 2; i < n; ++i) {
16            while((int) upper.size() >= 2) {
17                if(cross(upper.end()[-1] -
   ↪ upper.end()[-2], points[i] - upper.end()[-1]) >
   ↪ 0) {
18                    upper.pop_back();
19                } else {
20                    break;
21                }
22            }
23            upper.push_back(points[i]);
24        }
25        return upper;
26    };
27    vector<Point<T>> upper = build();
28    reverse(points.begin(), points.end());
29    vector<Point<T>> lower = build();
30    lower.pop_back();
31    upper.insert(upper.end(), lower.begin() + 1,
   ↪ lower.end());
32    return upper;
33 }

```
