ICPC NTHU valderyayaorz

llshang valderyaya Penguin07

September 14, 2022

Contents

1 Geometry

1.1 Point.h

```cpp
template<class T>
class Point {
public:
    T x, y;
    Point() : x(0), y(0) {}
    Point(const T& a, const T& b) : x(a), y(b) {}
    template<class U>
    explicit Point(const Point<U>& p) :
    ↪ x(static_cast<T>(p.x)), y(static_cast<T>(p.y))
    ↪ {}
    Point(const pair<T, T>& p) : x(p.first),
    ↪ y(p.second) {}
    Point(const complex<T>& p) : x(real(p)),
    ↪ y(imag(p)) {}
    explicit operator pair<T, T>() const {
        return pair<T, T>(x, y);
    }
    explicit operator complex<T>() const {
        return complex<T>(x, y);
    }
    inline Point& operator+=(const Point& rhs) {
        x += rhs.x;
        y += rhs.y;
        return *this;
    }
    inline Point& operator-=(const Point& rhs) {
        x -= rhs.x;
        y -= rhs.y;
        return *this;
    }
    inline Point& operator*=(const T& rhs) {
        x *= rhs;
        y *= rhs;
        return *this;
    }
    inline Point& operator/=(const T& rhs) {
        x /= rhs;
        y /= rhs;
        return *this;
    }
    template<class U>
    inline Point& operator+=(const Point<U>& rhs) {
        return *this += Point<T>(rhs);
    }
    template<class U>
    inline Point& operator-=(const Point<U>& rhs) {
        return *this -= Point<T>(rhs);
    }
    inline Point operator+() const {
        return *this;
    }
    inline Point operator-() const {
        return Point(-x, -y);
    }
    inline Point operator+(const Point& rhs) {
        return Point(*this) += rhs;
    }
    inline Point operator-(const Point& rhs) {
        return Point(*this) -= rhs;
    }
    inline Point operator*(const T& rhs) {
        return Point(*this) *= rhs;
    }
    inline Point operator/(const T& rhs) {
        return Point(*this) /= rhs;
    }
    inline bool operator==(const Point& rhs) {
        return x == rhs.x && y == rhs.y;
    }
    inline bool operator!=(const Point& rhs) {
        return !(*this == rhs);
    }
    inline T dist2() const {
        return x * x + y * y;
    }
    inline long double dist() const {
        return sqrt(dist2());
    }
    inline Point unit() const {
        return *this / this->dist();
    }
    inline long double angle() const {
        return atan2(y, x);
    }
    inline friend T dot(const Point& lhs, const
    ↪ Point& rhs) {
        return lhs.x * rhs.x + lhs.y * rhs.y;
    }
    inline friend T cross(const Point& lhs, const
    ↪ Point& rhs) {
        return lhs.x * rhs.y - lhs.y * rhs.x;
    }
    inline friend Point dot_cross(const Point& lhs,
    ↪ const Point& rhs) {
        return Point(dot(lhs, rhs), cross(lhs,
    ↪ rhs));
    }
```

```
};
template<class T>
istream& operator>>(istream& in, Point<T>& p) {
    return in >> p.x >> p.y;
}
```

1.2 ConvexHull.h

```
// @return the points of the convex hull in clock-wise order
template<class T>
vector<Point<T>> ConvexHull(vector<Point<T>> points) {
    const int n = (int) points.size();
    sort(points.begin(), points.end(), [](const Point<T>& a, const Point<T>& b) {
        if(a.x == b.x) {
            return a.y < b.y;
        }
        return a.x < b.x;
    });
    auto build = [&]() {
        vector<Point<T>> upper;
        upper.push_back(points[0]);
        upper.push_back(points[1]);
        for(int i = 2; i < n; ++i) {
            while((int) upper.size() >= 2) {
                if(cross(upper.end()[-1] - upper.end()[-2], points[i] - upper.end()[-1]) > 0) {
                    upper.pop_back();
                } else {
                    break;
                }
            }
            upper.push_back(points[i]);
        }
        return upper;
    };
    vector<Point<T>> upper = build();
    reverse(points.begin(), points.end());
    vector<Point<T>> lower = build();
    lower.pop_back();
    upper.insert(upper.end(), lower.begin() + 1, lower.end());
    return upper;
}
```