Advanced Algorithms for Programming Contests
Lecture 7. Advanced geometry

Bakhodir Ashirmatov,
Thilo Stier,
Philip Schär

University of Göttingen
Institute of Computer Science

05.06.2019

Overview

Point

- Vector
- Cross product
- Clockwise and counterclockwise turns
- Triangle area

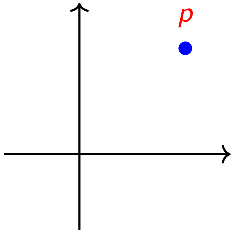
Polygon

- Triangle method for polygon area
- Trapeze method for polygon area
- Point inside a polygon
- Convex hull

Point

- An ordered pair (x, y)

```
struct PT {
    double x, y;

    PT operator+(const PT &p) const {
        return {x + p.x, y + p.y};
    }
    PT operator-(const PT &p) const {
        return {x - p.x, y - p.y};
    }
    PT operator*(double c) const {
        return {x * c, y * c};
    }
    PT operator/(double c) const {
        return {x / c, y / c};
    }
};
```



Vector

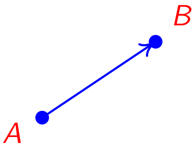
Vector

- $\vec{AB}$ is an object connecting the starting point A with the endpoint B

Vector length

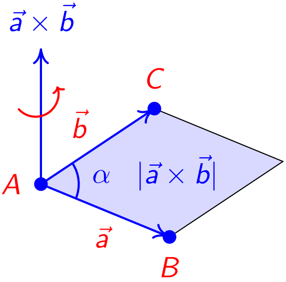
- $|\vec{AB}|$ is the distance between points A and B

```
PT ab = b - a;
```



Cross product

- A perpendicular vector to vectors $\vec{a}$ and $\vec{b}$ with
  - direction given by the right-hand rule
  - length equal to the area of the parallelogram



$$|\vec{a} \times \vec{b}| = |\vec{a}| \cdot |\vec{b}| \cdot \sin\alpha$$

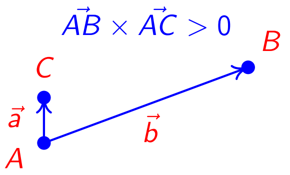
$$\vec{a} \times \vec{b} = \vec{a_x} \cdot \vec{b_y} - \vec{a_y} \cdot \vec{b_x}$$

$$\vec{AB} \times \vec{AC} = (B_x - A_x)(C_y - A_y) - (B_y - A_y)(C_x - A_x)$$

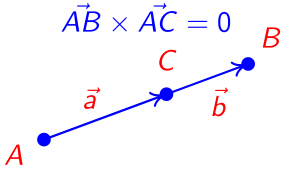
```
double cross(PT a, PT b) {
    return a.x * b.y - a.y * b.x;
}

double cross_ab_ac = cross(b - a, c - a);
```

Clockwise and counterclockwise turns

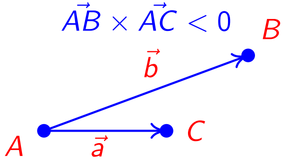
$$\vec{a} \times \vec{b} = \begin{cases} > 0 & \vec{a} \to \vec{b} \text{ is counterclockwise} \\ = 0 & \vec{a} \text{ and } \vec{b} \text{ are anti- or collinear} \\ < 0 & \vec{a} \to \vec{b} \text{ is clockwise} \end{cases}$$



$$\vec{AB} \times \vec{AC} = \begin{cases} > 0 & A \to B \to C \text{ is a left turn} \\ = 0 & C \text{ is on line } \vec{AB} \\ < 0 & A \to B \to C \text{ is a right turn} \end{cases}$$

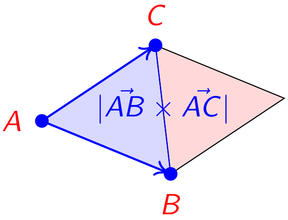



```
double cw(PT a, PT b) {
    return cross(a, b) < -EPS;
}

double ccw(PT a, PT b) {
    return cross(a, b) > EPS;
}

cw_ab_ac = cw(b - a, c - a);
```



Triangle area

$$S_{ABC} = \frac{|\vec{AB} \times \vec{AC}|}{2}$$

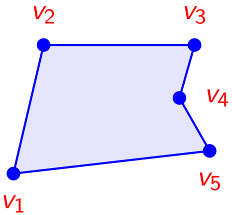



```
double triangle_area(PT a, PT b, PT c) {
    return abs(cross(b - a, c - a)) / 2;
}
```

Polygon

- A plane figure v bounded by a closed chain of segments (v_i, v_{i+1})

```
vector<PT> v;
```



Polygon area

Problem

- Given a polygon.
- Find the area of the polygon.

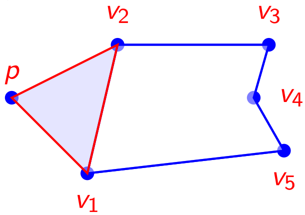
Triangle method for polygon area

Idea

- Divide the polygon into triangles.

$$S_v = \frac{\left| \sum_{i=1}^{N} \vec{pv_i} \times \vec{pv_{i+1}} \right|}{2}$$

```
double polygon_triangle_area(vector<PT> &v, PT p) {
    int n = (int) v.size();
    double sum = 0;
    for (int i = 0; i < n; ++i){
        int j = i + 1 < n ? i + 1 : 0;
        sum += cross(v[i] - p, v[j] - p);
    }
    return abs(sum) / 2;
}
```



Complexity: $O(N)$

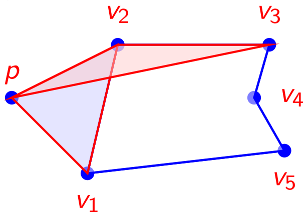
Triangle method for polygon area

Idea

- Divide the polygon into triangles.

$$S_v = \frac{\left|\sum_{i=1}^{N} \vec{pv_i} \times \vec{pv_{i+1}}\right|}{2}$$

```
double polygon_triangle_area(vector<PT> &v, PT p) {
    int n = (int) v.size();
    double sum = 0;
    for (int i = 0; i < n; ++i){
        int j = i + 1 < n ? i + 1 : 0;
        sum += cross(v[i] - p, v[j] - p);
    }
    return abs(sum) / 2;
}
```



Complexity: $O(N)$

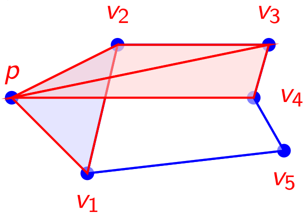
Triangle method for polygon area

Idea

- Divide the polygon into triangles.

$$S_v = \frac{\left|\sum_{i=1}^{N} \vec{pv_i} \times \vec{pv_{i+1}}\right|}{2}$$

```
double polygon_triangle_area(vector<PT> &v, PT p) {
    int n = (int) v.size();
    double sum = 0;
    for (int i = 0; i < n; ++i){
        int j = i + 1 < n ? i + 1 : 0;
        sum += cross(v[i] - p, v[j] - p);
    }
    return abs(sum) / 2;
}
```



Complexity: $O(N)$

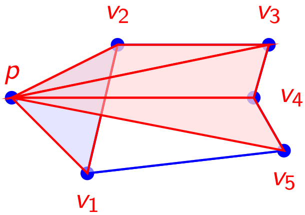
Triangle method for polygon area

Idea

- Divide the polygon into triangles.

$$S_v = \frac{\left|\sum_{i=1}^{N} \vec{pv_i} \times \vec{pv_{i+1}}\right|}{2}$$

```
double polygon_triangle_area(vector<PT> &v, PT p) {
    int n = (int) v.size();
    double sum = 0;
    for (int i = 0; i < n; ++i){
        int j = i + 1 < n ? i + 1 : 0;
        sum += cross(v[i] - p, v[j] - p);
    }
    return abs(sum) / 2;
}
```



Complexity: $O(N)$

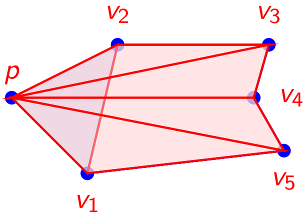
Triangle method for polygon area

Idea

- Divide the polygon into triangles.

$$S_v = \frac{\left| \sum_{i=1}^{N} \vec{pv_i} \times \vec{pv_{i+1}} \right|}{2}$$

```
double polygon_triangle_area(vector<PT> &v, PT p) {
    int n = (int) v.size();
    double sum = 0;
    for (int i = 0; i < n; ++i){
        int j = i + 1 < n ? i + 1 : 0;
        sum += cross(v[i] - p, v[j] - p);
    }
    return abs(sum) / 2;
}
```



Complexity: $O(N)$

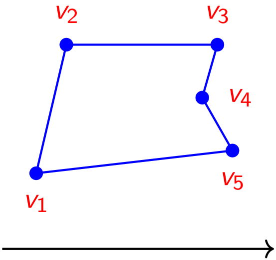
Trapeze method for polygon area

Idea

- Divide the polygon into trapezes.

$$S_v = \frac{|\sum(x_i - x_{i+1}) \cdot (y_i + y_{i+1})|}{2}$$

```
double polygon_trapeze_area(vector<PT> &v) {
    int n = (int) v.size();
    double sum = 0;
    for (int i = 0; i < n; ++i){
        int j = i + 1 < n ? i + 1 : 0;
        sum += (v[i].x - v[j].x) * (v[i].y + v[j].y);
    }
    return abs(sum) / 2;
}
```



Complexity: $O(N)$

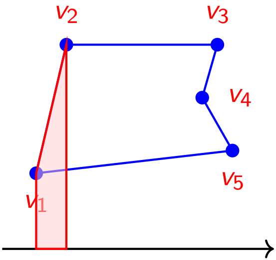
Trapeze method for polygon area

Idea

- Divide the polygon into trapezes.

$$S_v = \frac{|\sum(x_i - x_{i+1}) \cdot (y_i + y_{i+1})|}{2}$$

```
double polygon_trapeze_area(vector<PT> &v) {
    int n = (int) v.size();
    double sum = 0;
    for (int i = 0; i < n; ++i){
        int j = i + 1 < n ? i + 1 : 0;
        sum += (v[i].x - v[j].x) * (v[i].y + v[j].y);
    }
    return abs(sum) / 2;
}
```



Complexity: $O(N)$

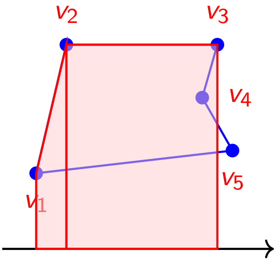
Trapeze method for polygon area

Idea

- Divide the polygon into trapezes.

$$S_v = \frac{|\sum(x_i - x_{i+1}) \cdot (y_i + y_{i+1})|}{2}$$

```
double polygon_trapeze_area(vector<PT> &v) {
    int n = (int) v.size();
    double sum = 0;
    for (int i = 0; i < n; ++i){
        int j = i + 1 < n ? i + 1 : 0;
        sum += (v[i].x - v[j].x) * (v[i].y + v[j].y);
    }
    return abs(sum) / 2;
}
```



Complexity: $O(N)$

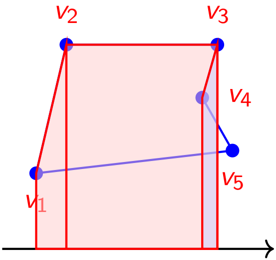
Trapeze method for polygon area

Idea

- Divide the polygon into trapezes.

$$S_v = \frac{|\sum(x_i - x_{i+1}) \cdot (y_i + y_{i+1})|}{2}$$

```
double polygon_trapeze_area(vector<PT> &v) {
    int n = (int) v.size();
    double sum = 0;
    for (int i = 0; i < n; ++i){
        int j = i + 1 < n ? i + 1 : 0;
        sum += (v[i].x - v[j].x) * (v[i].y + v[j].y);
    }
    return abs(sum) / 2;
}
```



Complexity: $O(N)$

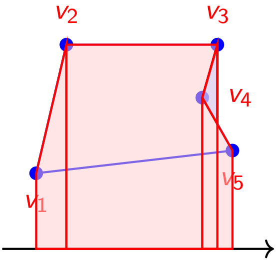
Trapeze method for polygon area

Idea

- Divide the polygon into trapezes.

$$S_v = \frac{|\sum (x_i - x_{i+1}) \cdot (y_i + y_{i+1})|}{2}$$

```
double polygon_trapeze_area(vector<PT> &v) {
    int n = (int) v.size();
    double sum = 0;
    for (int i = 0; i < n; ++i){
        int j = i + 1 < n ? i + 1 : 0;
        sum += (v[i].x - v[j].x) * (v[i].y + v[j].y);
    }
    return abs(sum) / 2;
}
```



Complexity: $O(N)$

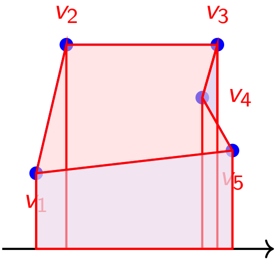
Trapeze method for polygon area

Idea

- Divide the polygon into trapezes.

$$S_v = \frac{|\sum(x_i - x_{i+1}) \cdot (y_i + y_{i+1})|}{2}$$

```
double polygon_trapeze_area(vector<PT> &v) {
    int n = (int) v.size();
    double sum = 0;
    for (int i = 0; i < n; ++i){
        int j = i + 1 < n ? i + 1 : 0;
        sum += (v[i].x - v[j].x) * (v[i].y + v[j].y);
    }
    return abs(sum) / 2;
}
```



Complexity: $O(N)$

Point inside a polygon

Problem

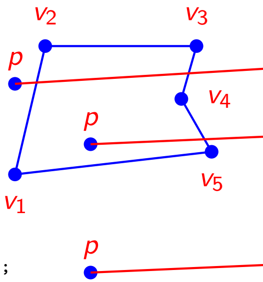
- Given a polygon and a point.
- Find out if the point is inside of the polygon.

Idea

- Cast a ray from the point and check the number of intersections with the polygon.
- If the number is odd, the point is inside.
- If the number is even, the point is outside.

Point inside a polygon

```
bool check_inside(vector<PT> &v, PT p) {
    int n = (int) v.size();
    for (int i = 0; i < n; ++i) {
        if (dis(v[i], p) < EPS) {
            return true;
        }
    }
    PT pinf = {p.x + INF, p.y + 1};
    bool cnt = 0;
    for (int i = 0; i < n; i++){
        int j = i + 1 < n ? i + 1 : 0
        cnt ^= intersect_segment_ray(v[i], v[j], p, pinf);
    }
    return cnt;
}
```



Complexity $O(N)$

Point inside a convex polygon

Problem

- Given a convex polygon and a point.
- Check efficiently, whether the point p is in the polygon.

Idea

- Find a point zero inside of the polygon.
- Sort the segments of the polygon by angle around zero.
- Find the angle of p around zero.
- If p is closer to zero than the segment at the angle, it is inside.
- Otherwise it is outside.

Point inside a convex polygon

- Represent angles as points.
- Compare by angle of point to positive x-axis

```
bool cmp_angle(const PT &v1, const PT &v2) {
    return v1.y > 0 && (v2.y > 0 && v1.x * v2.y > v2.x * v1.y ||
                        v2.y == 0 && v2.x < 0 ||
                        v2.y < 0) ||
           v1.y == 0 && (v1.x > 0 && (v2.y != 0 || v2.x < 0) ||
                         v1.x < 0 && v2.y < 0) ||
           v1.y < 0 && v2.y < 0 && v2.x * v1.y < v1.x * v2.y;
}
```

Point inside a convex polygon

- This function assumes, that the polygon is given counter-clockwise.

```
vector<PT> precalc(vector<PT> &v, PT &zero) {
    int n = v.size(), min_i = 0;
    vector<PT> a;
    for (int i = 0; i < n; i++) {
        a.push_back(v[i] - zero);
        if (cmp_angle(a[i], a[min_i]))
            min_i = i;
    }
    rotate(a.begin(), a.begin() + min_i, a.end());
    rotate(v.begin(), v.begin() + min_i, v.end());
    return a;
}
```

Complexity: $O(N)$

Point inside a convex polygon

```
bool point_inside_convex_polygon(vector<PT> &v, PT zero,
                                 vector<PT> a, PT &p) {
    int n = v.size();
    int ub = upper_bound(a.begin(), a.end(),
                         p - zero, cmp_angle) - a.begin();
    int i1, i2;
    if (ub == n || ub == 0)
        i1 = n-1, i2 = 0;
    else
        i1 = ub - 1, i2 = ub;
    return ccw(v[i2] - v[i1], p - v[i1]);
}
```

Complexity: $O(\log N)$

Convex hull

Problem

- Given N points.
- Find the convex hull of the points (which is a polygon).

Idea

- Sort points in lexicographical order.
- The first and last point have to be part of the convex hull.
- Build the lower and upper half of the convex hull separately:
    - Add new point to polygon.
    - Remove all points that are not needed anymore.
- Join the two halves.

Convex hull

```
bool cmp(PT a, PT b) {
    return a.x < b.x || a.x == b.x && a.y < b.y;
}

vector<PT> convex_hull(vector<PT> &v) {
    if (v.size() == 1) {
        return;
    }
    sort(v.begin(), v.end(), &cmp);
    PT p1 = v[0], p2 = v.back();
    vector<PT> up, down;
    up.push_back(p1);
    down.push_back(p1);
    int n = (int) v.size();
```

Convex hull

```
    for (int i = 1; i < n; ++i) {
        if (i == n - 1 || cw(v[i] - p1, p2 - v[i])) {
            while (up.size() >= 2 &&
                !cw(up[up.size()-1]-up[up.size()-2], v[i]-up[up.size()-1]))
                up.pop_back();
            up.push_back(v[i]);
        }
        if (i == n - 1 || ccw(v[i] - p1, p2 - v[i])) {
            while (down.size() >= 2 &&
                !ccw(down[down.size()-1]-down[down.size()-2],
                    v[i]-down[down.size()-1]))
                down.pop_back();
            down.push_back(v[i]);
        }
    }

    vector<PT> hull = up;
    for (int i = down.size() - 2; i > 0; --i) {
        hull.push_back(down[i]);
    }
    return hull;
}
```

Complexity $O(N \log N)$

Do your homework!