Points, Lines, and Vectors

CS 491 CAP

Dr. Mattox Beckman

University of Illinois at Urbana-Champaign
Department of Computer Science

Fall 2023

Objectives

- Identify some of the corner cases computational geometry problems have.
- Develop a strategy for dealing with geometry problems in a contest.
- Review basic formula for
  - Points
  - Lines
  - Vectors
- Most code samples from Competitive Programming 3.

- These problems can be tricky
  - Tedious coding
  - High probability of `WA` initially
- Edge cases!
  - What is lines are parallel?
  - Can the polygons be concave?
  - Check your assumptions!
- Strategy
  - Usually solve these last
  - Bring library code to the contest

Representing Integer Points

```
struct point_i {
   int x, y;
   point_i() { x = y = 0; }
   point_i(int _x, int _y) : x(_x), y(_y) {}
   bool operator=(point_i & other) const {
     return x == other.x && y == other.y;
   }
   bool operator<(point_i & other) const {
     if (x == other.x)
        return y < other.y;
     else return x < other.x;
  }
};
```

Representing Floating Points

```
#include <math.h>
#define EPS 1E-9

struct point {
   double x, y;
   point() { x = y = 0; }
   point(double _x, double _y) : x(_x), y(_y) {}
   bool operator=(point & other) const {
     return fabs(x - other.x) < EPS && fabs(y - other.y) <
   }
   bool operator<(point & other) const {
     if (fabs(x - other.x) < EPS)
        return y < other.y;
     else return x < other.x;
  }
};
```

Formulae

- Distance between two points

$$\sqrt{(a_x - b_x)^2 + (a_y - b_y)^2}$$

- Counter-clockwise rotation by θ

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

```
double hypot(point &a, point &b) const {
    double dx=a.x-b.x;
    double dy=a.y-b.y;

    return sqrt(dx * dx + dy * dy);
}
point ccw(point &a, double &theta) const {
    double st = sin(theta);
    double ct = cos(theta);
```

Representation

$$ax + by + c = 0$$

- if $b = 0$ the line is vertical.

```
// From Competitive Programming 3

struct line {
  point a, b, c;
};

void pointsToLine(point p1, point p2, line &l) {
  if (fabs(p1.x - p2.x) < EPS) {// vertical line
    l.a = 1.0; l.b = 0.0; l.c = -p1.x;
  } else {
    l.a = -(double)(p1.y - p2.y) / (p1.x - p2.x);
    l.b = 1.0; // IMPORTANT: we fix the value of b to 1.
    l.c = -(double)(l.a * p1.x) - p1.y;
} }
```

Parallel Lines

- Given lines $a_1x + b_1y + c_1$ and $a_2x + b_2y + c_2$
  - If $a_1 = a_2 \wedge b_1 = b_2$ the lines are parallel.
  - If also $c_1 = c_2$ the lines are identical.

```
bool areParallel(line l1, line l2) {
    return (fabs(l1.a-l2.a) < EPS) && (fabs(l1.b-l2.b) < EP
}
bool areSame(line l1, line l2) {
    return areParallel(l1 ,l2) && (fabs(l1.c - l2.c) < EPS)
}
```

Intersections

$$a_1x + b_1y + c_1 = 0$$
$$a_2x + b_2y + c_2 = 0$$

```
// returns true (+ intersection point) if two lines are int
bool areIntersect(line l1, line l2, point &p) {
    if (areParallel(l1, l2)) return false; // no intersect
    // solve system of 2 linear algebraic equations with 2
    p.x = (l2.b * l1.c - l1.b * l2.c) / (l2.a * l1.b - l1.a
    // special case: test for vertical line to avoid divis
    if (fabs(l1.b) > EPS) p.y = -(l1.a * p.x + l1.c);
      else p.y = -(l2.a * p.x + l2.c);
    return true;
}
```

Representation

- A vector represents a direction. Similar to a point, but different interpretation.

```
struct vec {
    double x, y;
    vec(double _x, double _y) : x(_x), y(_y) {}
};

vec toVec(point a, point b) {  // convert 2 points to vecto
    return vec(b.x - a.x, b.y - a.y);
}
vec scale(vec v, double s) { // nonnegative s = [<1 .. 1 ..
    return vec(v.x * s, v.y * s);
}
 // shorter.same.longer
point translate(point p, vec v) {// translate p according t
    return point(p.x + v.x , p.y + v.y);
}
```

Shortest Distance

- Dot product "multiplies" vectors.
- If zero, then the vectors are at right angles.
- The variable *u* will be between 0 to 1 if the intersection is between the points given.

```
double dot(vec a, vec b) {
   return (a.x * b.x + a.y * b.y);
}
double norm_sq(vec v) {
   return v.x * v.x + v.y * v.y;
}

// returns the distance from p to the line defined by two p
double distToLine(point p, point a, point b, point &c) {
   // formula: c = a + u * ab
   vec ap = toVec(a, p), ab = toVec(a, b);
   double u = dot(ap, ab) / norm_sq(ab);
   c = translate(a, scale(ab, u));
```

Shortest Distance: Line Segment

```
double distToLineSegment(point p, point a, point b, point &
    vec ap = toVec(a, p), ab = toVec(a, b);
    double u = dot(ap, ab) / norm_sq(ab);
    if (u < 0.0) {
        c = point(a.x, a.y); // closer to a
        return dist(p, a);
    }
     // Euclidean distance between p and a
    if (u > 1.0) {
        c = point(b.x, b.y); // closer to b
        return dist(p, b); }
    // Otherwise, do the normal thing
    c = translate(a, scale(ab, u));
    return dist(p, c);
}
```

Angles

- The angle between two lines induced by three points *aob*
- Dot product $oa \cdot ob = |oa| \times |ob| \times cos(\theta)$
- Solve for θ to get $\theta = arccos(oa \cdot ob/(|oa| \times |ob|))$

```
double angle(point a, point o, point b) { // returns angle
  vec oa = toVector(o, a), ob = toVector(o, b);
  return acos(dot(oa, ob) / sqrt(norm_sq(oa) * norm_sq(ob
}
```

Cross Products

- Given a line p, q and point r
- Let a be the vector pq and b be the vector pr
- Cross product $a \times b = a.x \times b.y - a.y \times b.x$
  - Magnitude is area of parallelogram
  - Positive means $p \to q \to r$ is a left turn.
  - Zero means p, q, r are colinear.
  - Negative means $p \to q \to r$ is a right turn.

```
// returns true if point r is on the left side of line pq
bool ccw(point p, point q, point r) {
  return cross(toVec(p, q), toVec(p, r)) > EPS;
}
// returns true if point r is on the same line as the line
bool collinear(point p, point q, point r) {
  return fabs(cross(toVec(p, q), toVec(p, r))) < EPS;
}
```