Geometry

Zachary Friggstad

Programming Club Meeting

Points

```
#include <complex>
typedef complex<double> point;

point p(1.0, 5.7);
p.real(); // x component
p.imag(); // y component
```

Points

```
#include <complex>
typedef complex<double> point;

point p(1.0, 5.7);
p.real(); // x component
p.imag(); // y component
```

Helpful operations

```
point p, q;
p+q; // exactly what you expect
p *= polar(1, theta); //rotate around (0,0) by theta radians
abs(p-q); //distance between p and q
arg(p); //angle from positive x-axis, measured in (-pi, pi]
```

Points

```
#include <complex>
typedef complex<double> point;

point p(1.0, 5.7);
p.real(); // x component
p.imag(); // y component
```

Helpful operations

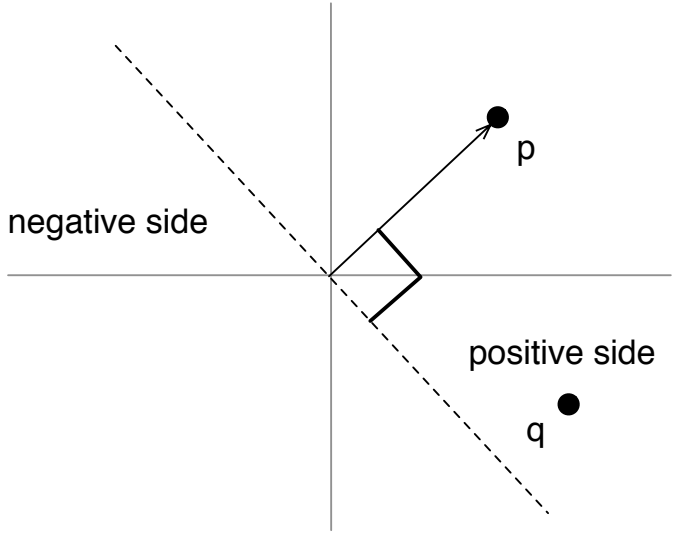
```
point p, q;
p+q; // exactly what you expect
p *= polar(1, theta); //rotate around (0,0) by theta radians
abs(p-q); //distance between p and q
arg(p); //angle from positive x-axis, measured in (-pi, pi]
```

One Annoyance: we can't update the x and y components without creating a new point.

Dot Product

$$p \circ q = p_x \cdot q_x + p_y \cdot q_y = ||p|| \cdot ||q|| \cdot \cos\theta$$

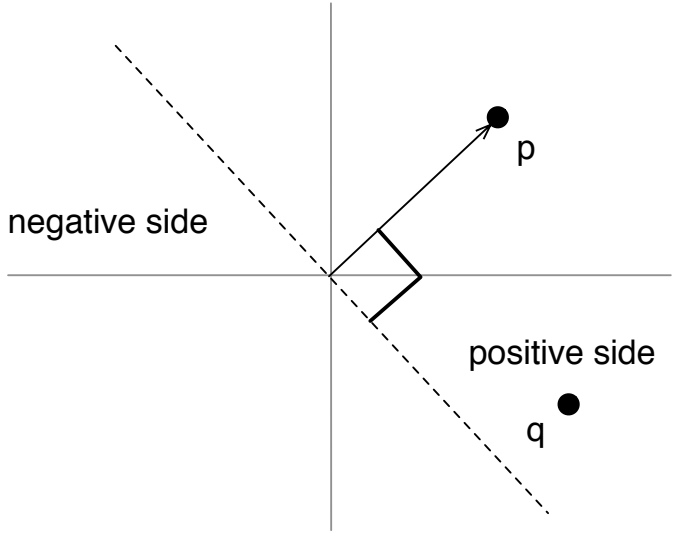
```
double dot(point p, point q) { return real(p*conj(q)); }
```



Dot Product

$$p \circ q = p_x \cdot q_x + p_y \cdot q_y = ||p|| \cdot ||q|| \cdot \cos\theta$$

```
double dot(point p, point q) { return real(p*conj(q)); }
```

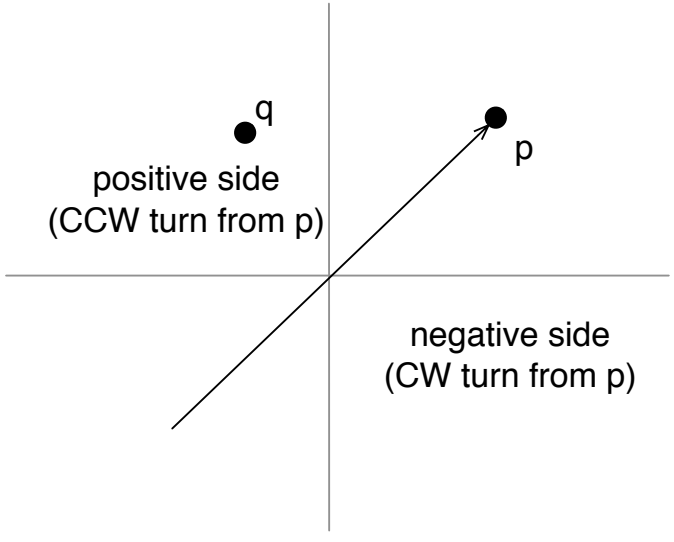


- $p \circ q = 0$ if $p \perp q$ (or one of them is 0)
- $p \circ p = ||p||^2$
- $p \circ q > 0$: q lies in the *same* general direction as p
- $p \circ q < 0$: q lies in the *opposite* general direction as p

Cross Product

$$p \times q = p_x \cdot q_y - p_y \cdot q_x = ||p|| \cdot ||q|| \cdot \sin\theta$$

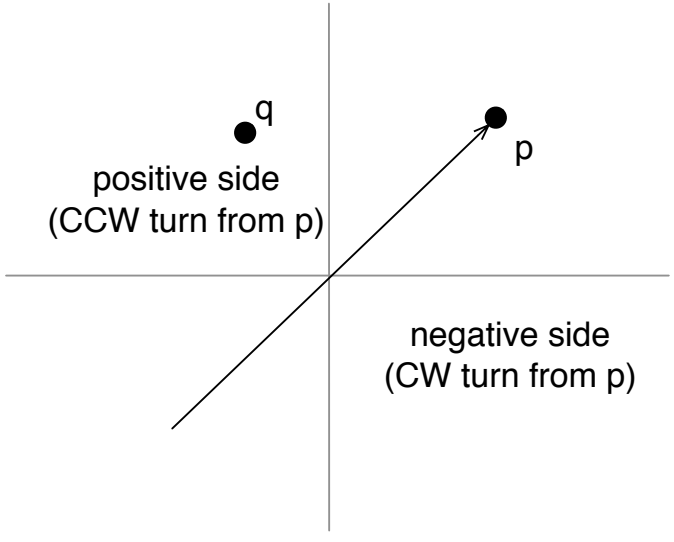
```
double cross(point p, point q) { return imag(p*conj(q)); }
```



Cross Product

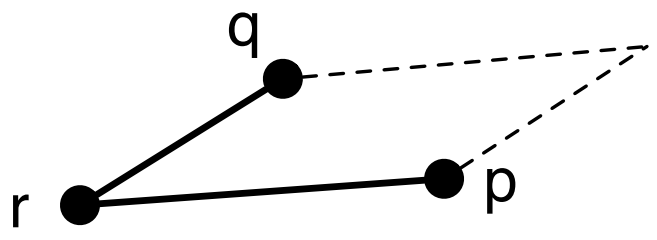
$p \times q = p_x \cdot q_y - p_y \cdot q_x = ||p|| \cdot ||q|| \cdot \sin\theta$

```
double cross(point p, point q) { return imag(p*conj(q)); }
```

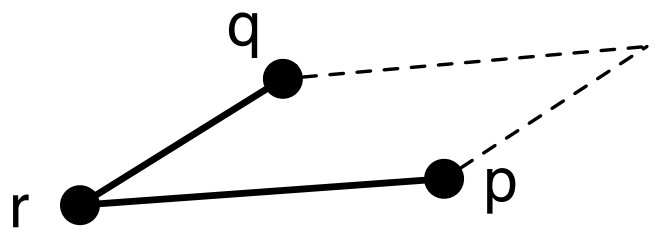


- $p \times q = 0$ if p and q are collinear with 0 (or one of them is 0)
- $p \times q > 0$: $\angle(p, q)$ is *counterclockwise*
- $p \times q < 0$: $\angle(p, q)$ is *clockwise*

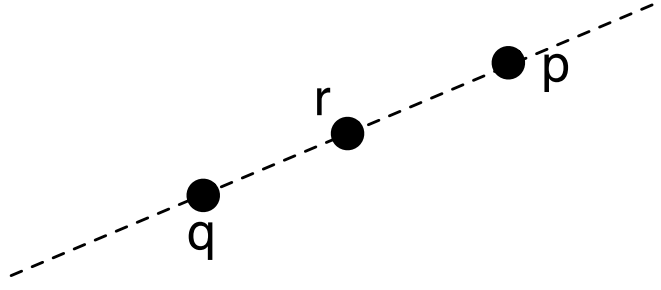
$\frac{1}{2}|(p-r)\times(q-r)|$ is the area of the triangle with corners p, q, r.



$\frac{1}{2}|(p-r)\times(q-r)|$ is the area of the triangle with corners p, q, r.



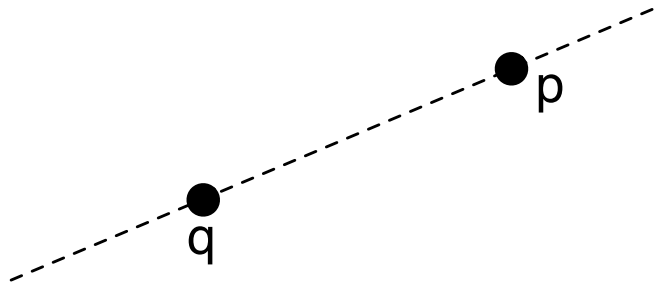
Tip: p lies on the line passing through points q and r **if and only if** $(q-p)\times(r-p)=0$.



Additionally: if p lies on this line then it lies between q and r **if and only** if $(q-p)\circ(r-p)\leq 0$.

Parametric representation of a line passing through p and q:

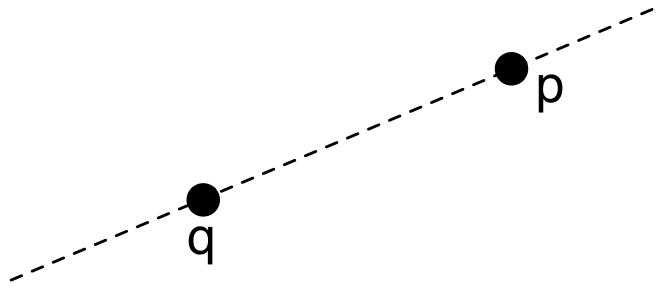
$$r(t) = (1 - t) \cdot p + t \cdot q, \quad t \in \mathbb{R}$$



Note $r(0) = p$ and $r(1) = q$.

Parametric representation of a line passing through p and q:

$$r(t) = (1 - t) \cdot p + t \cdot q, \quad t \in \mathbb{R}$$



Note $r(0) = p$ and $r(1) = q$.

The line segment connecting p and q is parameterized this way, with the restriction $0 \leq t \leq 1$.

Given two lines passing through p, q and p', q' respectively, compute their intersection point (if any).

Solve $(1 - t) \cdot p + t \cdot q = (1 - t')p' + t' \cdot q'$ for $t, t' \in \mathbb{R}$.

Given two lines passing through p, q and p', q' respectively, compute their intersection point (if any).

Solve $(1-t)\cdot p + t\cdot q = (1-t')p' + t'\cdot q'$ for $t, t' \in \mathbb{R}$.

$$\begin{bmatrix} q_x - p_x & p'_x - q'_x \\ q_y - p_y & p'_y - q'_y \end{bmatrix} \cdot \begin{bmatrix} t \\ t' \end{bmatrix} = \begin{bmatrix} p'_x - p_x \\ p'_y - p_y \end{bmatrix}$$

Given two lines passing through p, q and p', q' respectively, compute their intersection point (if any).

Solve $(1-t)\cdot p + t\cdot q = (1-t')p' + t'\cdot q'$ for $t, t' \in \mathbb{R}$.

$$\begin{bmatrix} q_x - p_x & p'_x - q'_x \\ q_y - p_y & p'_y - q'_y \end{bmatrix} \cdot \begin{bmatrix} t \\ t' \end{bmatrix} = \begin{bmatrix} p'_x - p_x \\ p'_y - p_y \end{bmatrix}$$

Use Cramer's rule: for $A \cdot x = b$ we have

$$x_i = \frac{\det A_i}{\det A}$$

Where A_i is the matrix obtained by replacing column i of A with b.

Given two lines passing through p, q and p', q' respectively, compute their intersection point (if any).

Solve $(1-t)\cdot p + t\cdot q = (1-t')p' + t'\cdot q'$ for $t, t' \in \mathbb{R}$.

$$\begin{bmatrix} q_x - p_x & p'_x - q'_x \\ q_y - p_y & p'_y - q'_y \end{bmatrix} \cdot \begin{bmatrix} t \\ t' \end{bmatrix} = \begin{bmatrix} p'_x - p_x \\ p'_y - p_y \end{bmatrix}$$

Use Cramer's rule: for $A \cdot x = b$ we have

$$x_i = \frac{\det A_i}{\det A}$$

Where A_i is the matrix obtained by replacing column i of A with b.

Tip: $\det \begin{bmatrix} a & b \\ c & d \end{bmatrix} = ad - bc$.

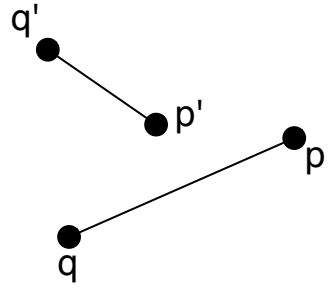
$\det A = 0$ means the lines are parallel.

Code

Note the 2×2 determinants can just be replaced by cross products!

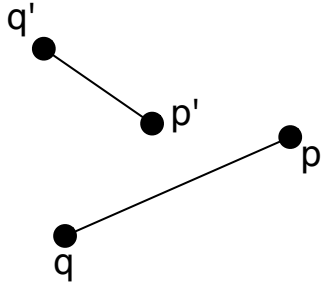
```
// find where lines p1q1 and p2q2 intersect
// assumes p1 != q1, p2 != q2
point isect(point p1, point q1, point p2, point q2) {
    double den = cross(q1-p1, p2-q2);
    if (fabs(den) < EPS) {
        // parallel lines, do whatever you want
        // they coincide iff fabs(cross(p1-p2, q1-p2)) < 1e-8
    }
    double t = cross(p2-p1, p2-q2)/den;

    return (1-t)*p1 + t*q1;
}
```

Computing the intersection of two line segments.



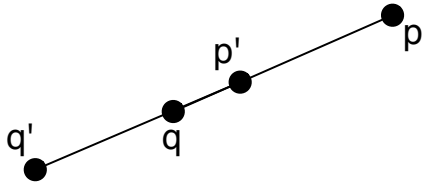
- If the infinite lines intersect (i.e. not parallel), get values t, t'.
- Ensure $0 \leq t \leq 1$ and $0 \leq t' \leq 1$.

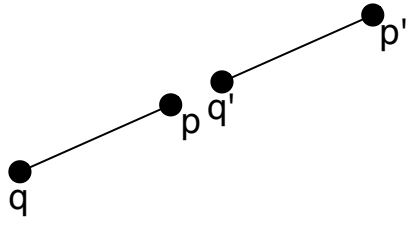
Computing the intersection of two line segments.



- If the infinite lines intersect (i.e. not parallel), get values t, t'.
- Ensure $0 \leq t \leq 1$ and $0 \leq t' \leq 1$.

If lines are colinear:

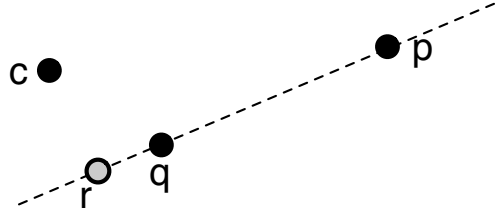




- If $p = p'$ and $q = q'$ or vice versa, they overlap.
- Else if a point is in the interior of the other segment, they overlap.
- Else if the segments share a point, they touch only at that point.
- Else they do not touch.

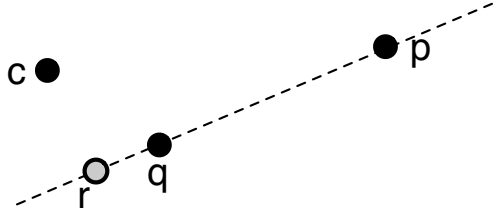
Closest point on a line.

Given a line passing through p, q and another point c, find the point r on the line $\overline{pq}$ nearest to c.



Closest point on a line.

Given a line passing through p, q and another point c, find the point r on the line $\overline{pq}$ nearest to c.

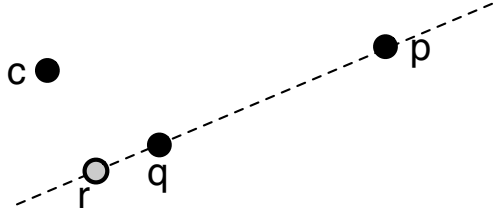


Assume $c = (0, 0)$ by shifting if necessary.

Minimize $||(1 - t)p + t \cdot q||^2$ over $t \in \mathbb{R}$ (avoids the square root).

Closest point on a line.

Given a line passing through p, q and another point c, find the point r on the line $\overline{pq}$ nearest to c.



Assume $c = (0, 0)$ by shifting if necessary.

Minimize $||(1 - t)p + t \cdot q||^2$ over $t \in \mathbb{R}$ (avoids the square root).

This is just minimizing some quadratic:

$$A \cdot t^2 + B \cdot t + C$$

Tip: the minimum is at $-\frac{B}{2A}$. If $A = 0$, then $p = q$.

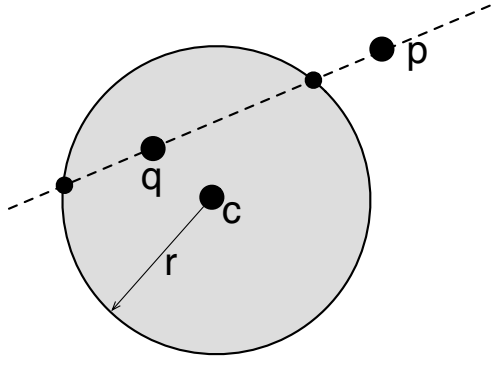
Code

After working out details...

```
// return the point on the line pq closest to c
point closest_point(point p, point q, point c) {
    p -= c; q -= c;
    double den = norm(p-q);
    if (fabs(den) < 1e-8) {
        // p=q, do whatever you want
    }
    double t = dot(p, p-q)/den;
    return (1-t)*p + t*q + c;
}
```

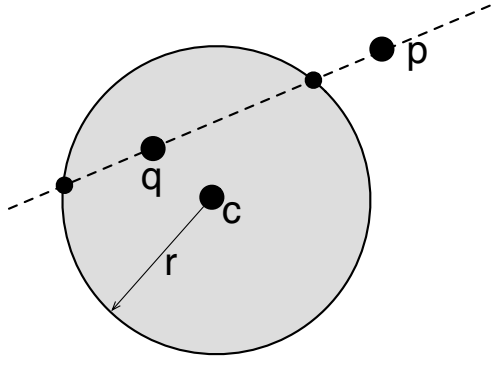
Line-Circle Intersection

A line $\overline{pq}$ and a circle (r, c) (radius/center point) does the line puncture the circle?



Line-Circle Intersection

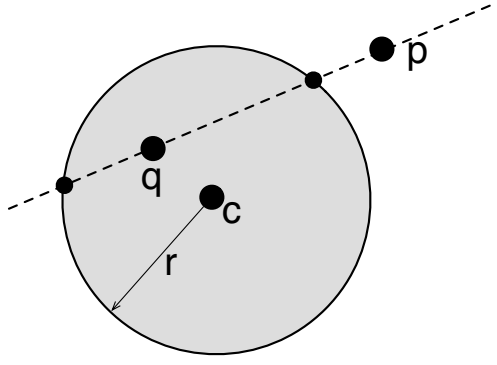
A line $\overline{pq}$ and a circle (r, c) (radius/center point) does the line puncture the circle?



Assume $c = 0$ by shifting and solve $||(1 - t) \cdot p + t \cdot q||^2 = r^2$ for t.

Line-Circle Intersection

A line $\overline{pq}$ and a circle (r, c) (radius/center point) does the line puncture the circle?



Assume $c = 0$ by shifting and solve $||(1-t)\cdot p + t\cdot q||^2 = r^2$ for t.

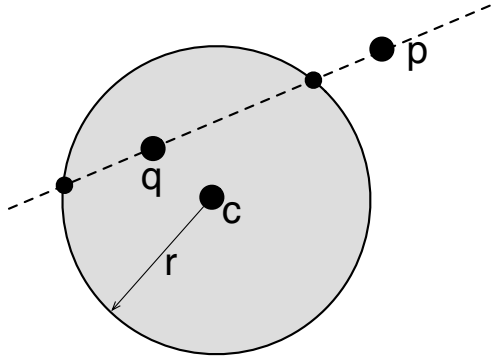
Rearrange: $A\cdot t^2 + B\cdot t + C = 0$. $A = 0$ means $p = q$.

Tip: Use the quadratic formula

$$\frac{-B \pm \sqrt{\Delta}}{2A} \quad \text{where} \quad \Delta = B^2 - 4AC.$$

Line-Circle Intersection

A line $\overline{pq}$ and a circle (r, c) (radius/center point) does the line puncture the circle?



Assume $c = 0$ by shifting and solve $||(1-t)\cdot p + t\cdot q||^2 = r^2$ for t.

Rearrange: $A\cdot t^2 + B\cdot t + C = 0$. $A = 0$ means $p = q$.

Tip: Use the quadratic formula

$$\frac{-B \pm \sqrt{\Delta}}{2A} \quad \text{where} \quad \Delta = B^2 - 4AC.$$

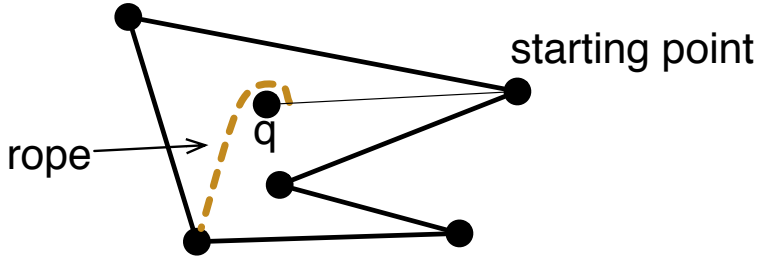
$\Delta < 0$: line misses, $\Delta = 0$: line tangent, $\Delta > 0$: line punctures.

Code

```
// return # of intersections and set parameters
// z1, z2 to these points (if applicable)
int line_circle(point c, double r, point p, point q,
                point& z1, point& z2) {
    p -= c; q -= c;
    double A = norm(p-q), B = 2*dot(p, q-p), C = norm(p)-r*r;
    double disc = B*B - 4*A*C;

    if (disc + EPS < 0) return 0;

    double rt = sqrt(fabs(disc));
    double t1 = (-B+rt)/(2*A);
    double t2 = (-B-rt)/(2*A);

    z1 = (1-t1)*p + t1*q + c;
    z2 = (1-t2)*p + t2*q + c;
    return (disc < EPS ? 1 : 2);
}
```

Point q in Polygon $P : p_1, \ldots, p_n$?

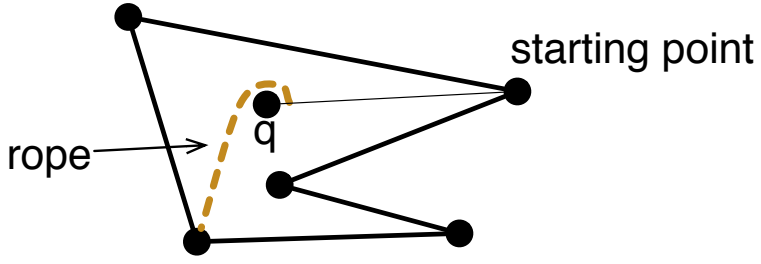
Intuition: walk about the polygon with a rope taut with "post" q.



If q is inside, the rope wraps around once. If q is outside, the rope doesn't wrap around.

Point q in Polygon $P : p_1, \ldots, p_n$?

Intuition: walk about the polygon with a rope taut with "post" q.

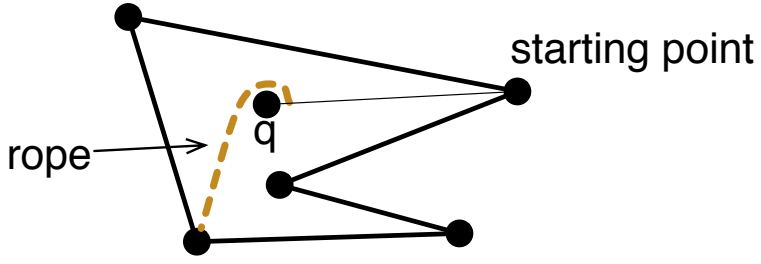


If q is inside, the rope wraps around once. If q is outside, the rope doesn't wrap around.

```
double delta = 0;
for (int i = 0, j = n-1; i < n; j = i++)
    delta += arg((p[i]-q) / (p[j]-q)); //change in angle
return fabs(delta) > 1; //|delta| is 0 or 2*pi
```

Point q in Polygon $P : p_1, \ldots, p_n$?

Intuition: walk about the polygon with a rope taut with "post" q.

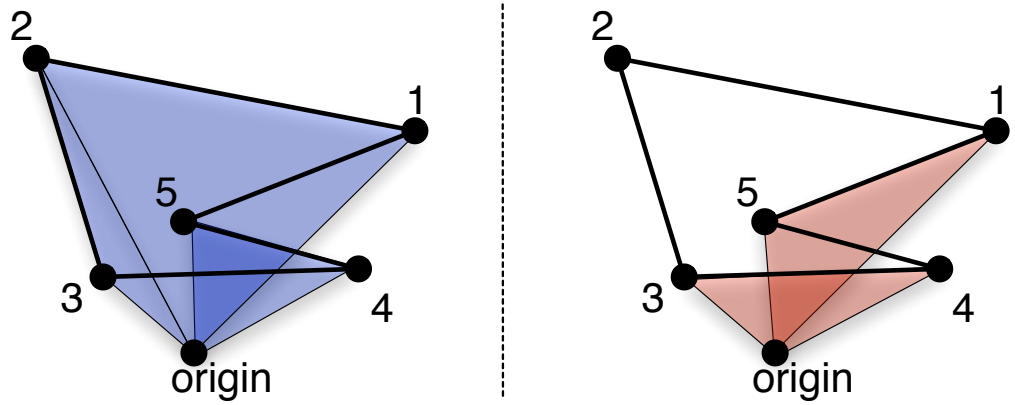


If q is inside, the rope wraps around once. If q is outside, the rope doesn't wrap around.

```
double delta = 0;
for (int i = 0, j = n-1; i < n; j = i++)
    delta += arg((p[i]-q) / (p[j]-q)); //change in angle
return fabs(delta) > 1; //|delta| is 0 or 2*pi
```

Have to add a bit more to check if q lies on the boundary (cross and dot product test mentioned earlier).

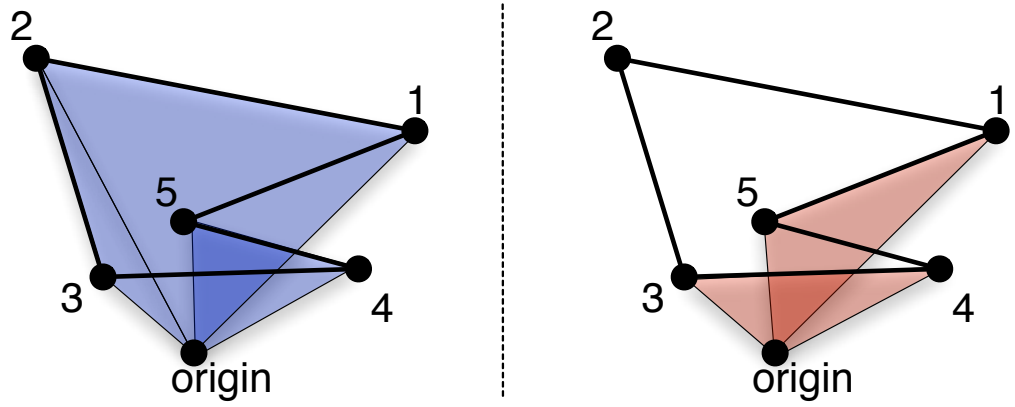
Area of a Polygon $P : p_1, \ldots, p_n$



blue triangle: count the area **positively** (CCW turn about origin)

red triangle: count the area **negatively** (CW turn about origin)

Area of a Polygon $P : p_1, \ldots, p_n$

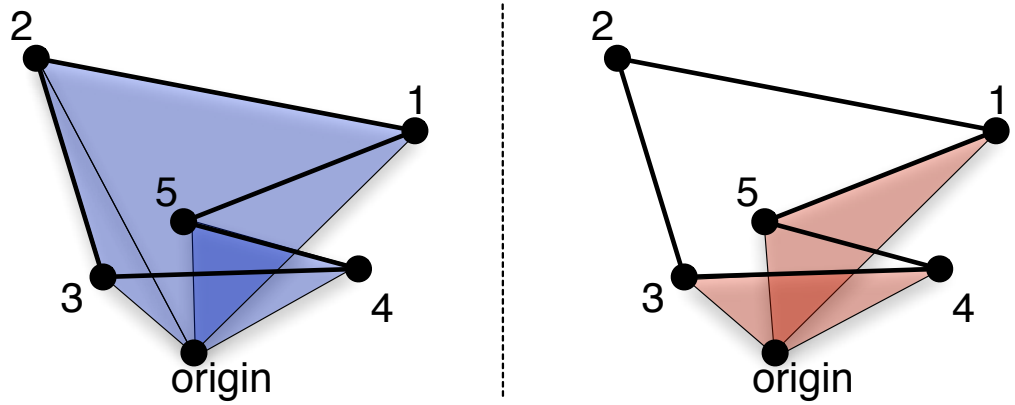


blue triangle: count the area **positively** (CCW turn about origin)

red triangle: count the area **negatively** (CW turn about origin)

```
double area = 0;
for (int i = 0, j = n-1; i < n; j = i++)
    //signed area of triangle [origin, p[i-1], p[i]]
    area += cross(p[i], p[j])*0.5;
return fabs(area);
```

Area of a Polygon $P : p_1, \ldots, p_n$



blue triangle: count the area **positively** (CCW turn about origin)

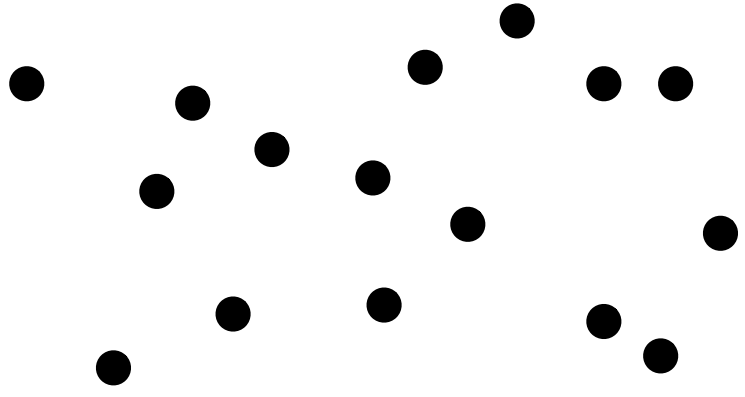
red triangle: count the area **negatively** (CW turn about origin)

```
double area = 0;
for (int i = 0, j = n-1; i < n; j = i++)
    //signed area of triangle [origin, p[i-1], p[i]]
    area += cross(p[i], p[j])*0.5;
return fabs(area);
```

Points outside are "cancelled", points inside are counted $+1$ times (after cancelling). The net sum is the area of P.

Convex Hull

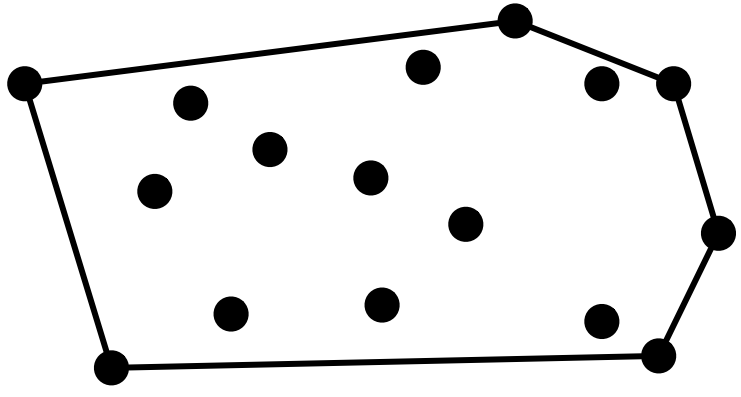
Given points $p_1, \ldots, p_n$, what is their *convex hull*?



Output: The points on the convex hull in CCW order.

Convex Hull

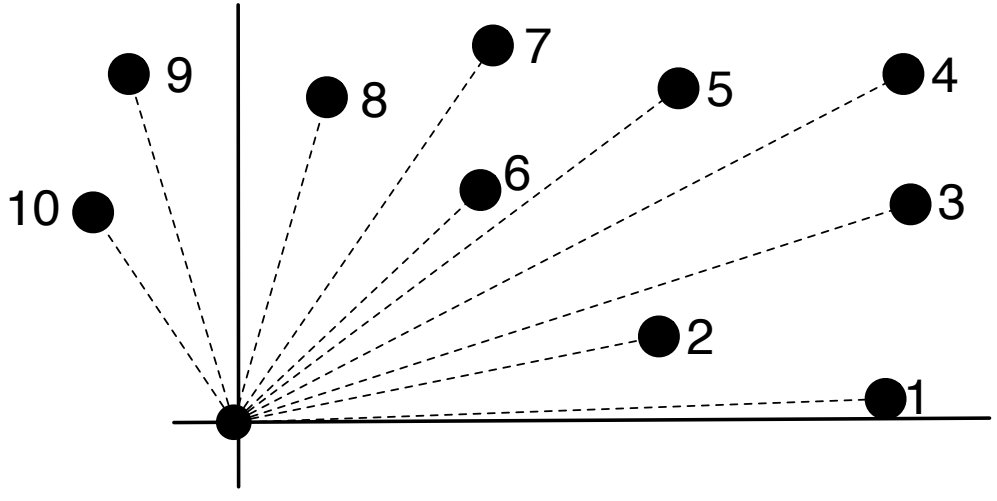
Given points $p_1, \ldots, p_n$, what is their *convex hull*?



Output: The points on the convex hull in CCW order.

Convex Hull

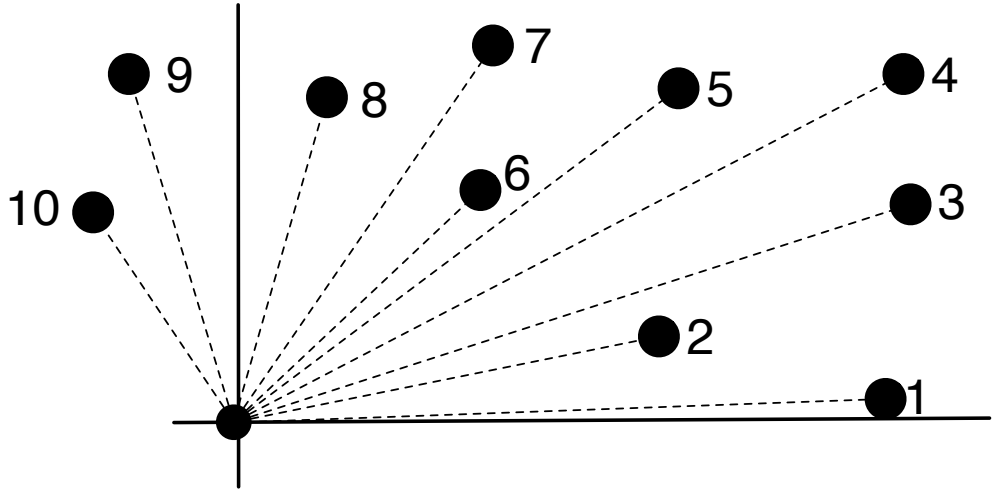
The bottom-most point must be on the convex hull. If there are many, choose the leftmost. Suppose this point is $(0, 0)$ by shifting all points if needed.



Sort all other points by their angle with the positive x-axis.
To check if $p_i < p_j$ according to this order:

Convex Hull

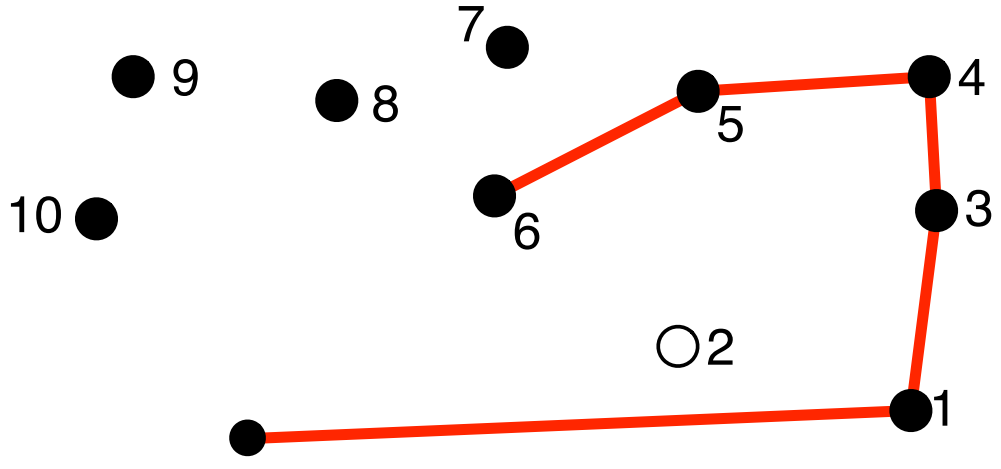
The bottom-most point must be on the convex hull. If there are many, choose the leftmost. Suppose this point is $(0, 0)$ by shifting all points if needed.



Sort all other points by their angle with the positive x-axis.
To check if $p_i < p_j$ according to this order: check $p_i \times p_j > 0$.

The origin and the first point after sorting are on the hull for sure.

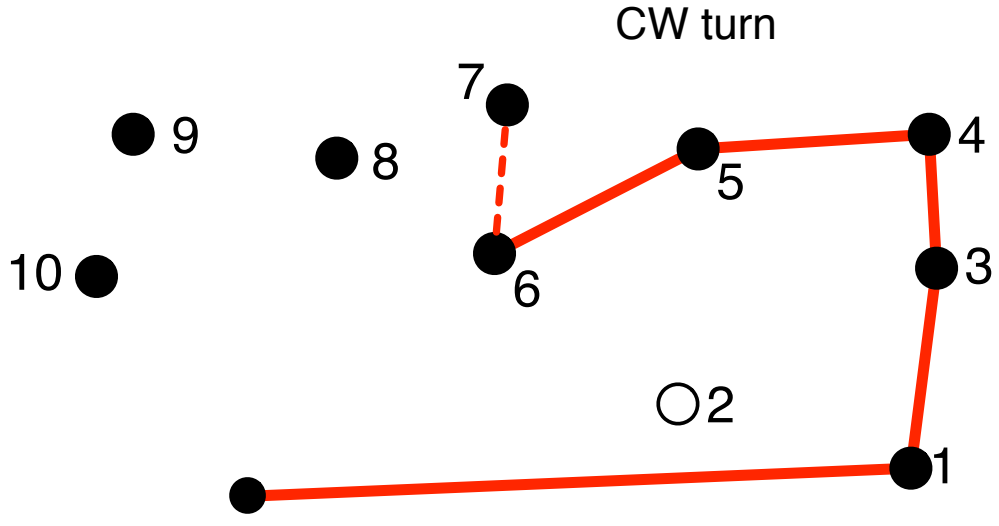
Build the hull one point at a time: add the points in the sorted order.



If one creates a clockwise turn with the previous two on the hull, then pop the end of the current hull.

The origin and the first point after sorting are on the hull for sure.

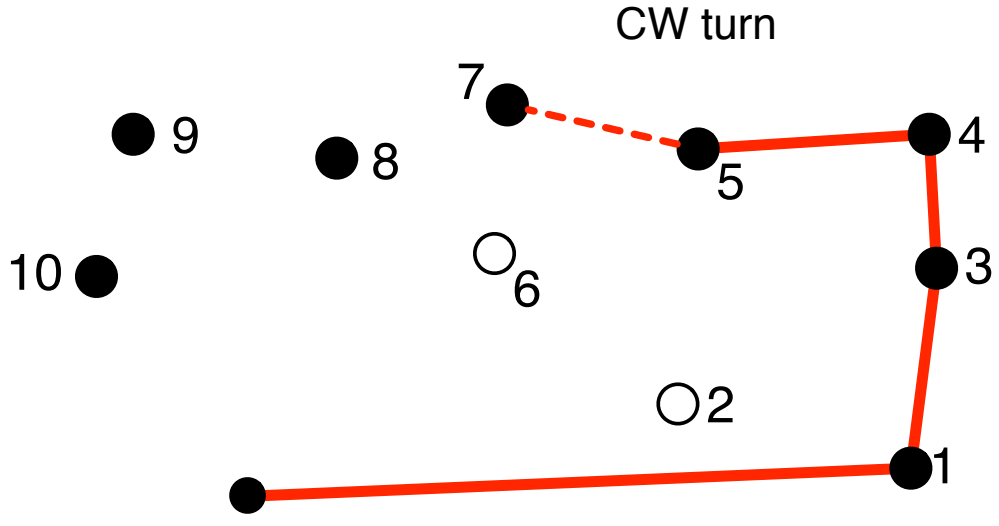
Build the hull one point at a time: add the points in the sorted order.



If one creates a clockwise turn with the previous two on the hull, then pop the end of the current hull.

The origin and the first point after sorting are on the hull for sure.

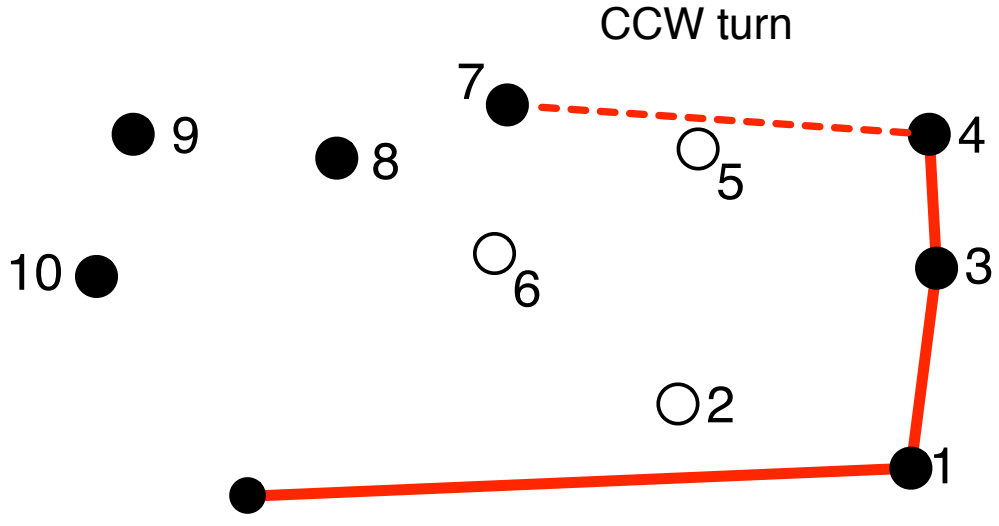
Build the hull one point at a time: add the points in the sorted order.



If one creates a clockwise turn with the previous two on the hull, then pop the end of the current hull.

The origin and the first point after sorting are on the hull for sure.

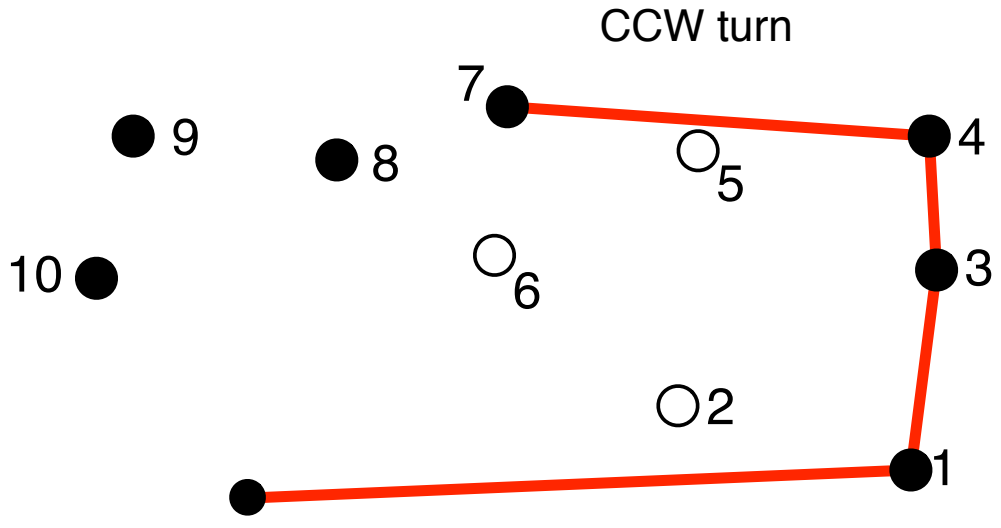
Build the hull one point at a time: add the points in the sorted order.



If one creates a clockwise turn with the previous two on the hull, then pop the end of the current hull.

The origin and the first point after sorting are on the hull for sure.

Build up the hull one point at a time: add the points in the sorted order.



If one creates a clockwise turn with the previous two on the hull, then pop the end of the current hull.

```
bool cmp(point a, point b) { return cross(a, b) > 0; }

point p[MAXN], hull[MAXN];
//Assume p[n-1] is the lowest point (breaking ties by
//taking leftmost) and no 3 points are collinear

for (int i = 0; i < n-1; ++i) p[i] -= p[n-1];
sort(p, p+n-1, cmp); //sort by angle from x-axis

hull[0] = point(0,0);
hull[1] = p[0];
int hs = 2;

for (int i = 1; i < n-1; ++i) {
  while (cross(hull[hs-1]-hull[hs-2],p[i]-hull[hs-2]) < 0)
    --hs;
  hull[hs++] = p[i];
}
for (int i = 0; i < hs; ++i) hull[i] += p[n-1];
```

Running time: $O(n \log n)$

Tips

Never use `float`, not enough precision. Use `double`.

Tips

Never use `float`, not enough precision. Use `double`.

```
const double PI = acos(-1.0);
```

Tips

Never use `float`, not enough precision. Use `double`.

```
const double PI = acos(-1.0);
```

Sometimes values that should be equal aren't quite due to floating point errors.

Tips

Never use `float`, not enough precision. Use `double`.

```
const double PI = acos(-1.0);
```

Sometimes values that should be equal aren't quite due to floating point errors.

Safer comparisons.

```
#define EPS 1e-8
double a, b;
if (fabs(a-b) < EPS) ...   //check if a == b
if (a + EPS < b) ...       //check if a < b
if (a < b + EPS) ...       //check if a <= b
```

Tips

Never use `float`, not enough precision. Use `double`.

```
const double PI = acos(-1.0);
```

Sometimes values that should be equal aren't quite due to floating point errors.

Safer comparisons.

```
#define EPS 1e-8
double a, b;
if (fabs(a-b) < EPS) ...   //check if a == b
if (a + EPS < b) ...       //check if a < b
if (a < b + EPS) ...       //check if a <= b
```

Can there be 3 collinear points? Can points be equal? How can you deal with this?

Missing Topics

`Voronoi diagrams/Delaunay triangulations, 3D convex hull, closest pair of points, furthest pair of points, triangulating a polygon.`

Next Time

Number Theory

Some Topics To Come

- Combinatorics and Arithmetic
- String Processing
- Matchings and Network Flow
- Probability Theory

Open Kattis - polygonarea

https://open.kattis.com/problems/polygonarea

Open Kattis - `polygonarea`

`https://open.kattis.com/problems/polygonarea`

- We already talked about how to compute the area of a polygon.

Open Kattis - `polygonarea`

`https://open.kattis.com/problems/polygonarea`

- We already talked about how to compute the area of a polygon.
- How can you check if it is in CW or CCW order?

Open Kattis - `polygonarea`

`https://open.kattis.com/problems/polygonarea`

- We already talked about how to compute the area of a polygon.
- How can you check if it is in CW or CCW order?
- Solution: check if the area is positive before returning the absolute value: > 0 means CCW, < 0 means CW.

Open Kattis - rafting

https://open.kattis.com/problems/wrapping

Open Kattis - `rafting`

`https://open.kattis.com/problems/wrapping`

- Think: what can we say about the polygon at the tightest squeeze point of the ride.

Open Kattis - `rafting`

`https://open.kattis.com/problems/wrapping`

- Think: what can we say about the polygon at the tightest squeeze point of the ride.
- Will there be a vertex of a polygon at the tightest squeeze?

Open Kattis - `rafting`

`https://open.kattis.com/problems/wrapping`

- Think: what can we say about the polygon at the tightest squeeze point of the ride.
- Will there be a vertex of a polygon at the tightest squeeze?
- For each vertex, find the shortest distance to all other line segments on the other polygon (do for both polygons).

Running time:

Open Kattis - `rafting`

`https://open.kattis.com/problems/wrapping`

- Think: what can we say about the polygon at the tightest squeeze point of the ride.
- Will there be a vertex of a polygon at the tightest squeeze?
- For each vertex, find the shortest distance to all other line segments on the other polygon (do for both polygons).

Running time: $O(n^2)$.

Open Kattis - wrapping

https://open.kattis.com/problems/wrapping

Open Kattis - `wrapping`

`https://open.kattis.com/problems/wrapping`

Basic approach:

- The total area of the objects is simple, you know the width and height of each (rectangle area $= w \cdot h$).

Open Kattis - `wrapping`

`https://open.kattis.com/problems/wrapping`

Basic approach:

- The total area of the objects is simple, you know the width and height of each (rectangle area $= w \cdot h$).
- You need the actual corners to compute the convex hull. Rotate the four corners of the axis-parallel rectangles. by the given angle: simple using complex points (recall: $\mathit{radians} = \frac{\pi}{180} \cdot \mathit{degrees}$).

Open Kattis - `wrapping`

`https://open.kattis.com/problems/wrapping`

Basic approach:

- The total area of the objects is simple, you know the width and height of each (rectangle area $= w \cdot h$).
- You need the actual corners to compute the convex hull. Rotate the four corners of the axis-parallel rectangles. by the given angle: simple using complex points (recall: $\mathit{radians} = \frac{\pi}{180} \cdot \mathit{degrees}$).
- Find the convex hull.

Open Kattis - `wrapping`

`https://open.kattis.com/problems/wrapping`

Basic approach:

- The total area of the objects is simple, you know the width and height of each (rectangle area $= w \cdot h$).
- You need the actual corners to compute the convex hull. Rotate the four corners of the axis-parallel rectangles. by the given angle: simple using complex points (recall: $\mathit{radians} = \frac{\pi}{180} \cdot \mathit{degrees}$).
- Find the convex hull.
- Compute the area of the convex hull using the algorithm discussed earlier.

Running time:

Open Kattis - `wrapping`

`https://open.kattis.com/problems/wrapping`

Basic approach:

- The total area of the objects is simple, you know the width and height of each (rectangle area $= w \cdot h$).
- You need the actual corners to compute the convex hull. Rotate the four corners of the axis-parallel rectangles. by the given angle: simple using complex points (recall: $\mathit{radians} = \frac{\pi}{180} \cdot \mathit{degrees}$).
- Find the convex hull.
- Compute the area of the convex hull using the algorithm discussed earlier.

Running time: $O(n \log n)$

Open Kattis - goofy

https://open.kattis.com/problems/wrapping

Open Kattis - goofy

`https://open.kattis.com/problems/wrapping`

Basic approach:

- Consider the line L passing through the given point p and point y (a parameter) on the vertical axis.

Open Kattis - goofy

`https://open.kattis.com/problems/wrapping`

Basic approach:

- Consider the line L passing through the given point p and point y (a parameter) on the vertical axis.
- Use the quadratic expression $||(1-t)\cdot(0,y)+t\cdot p||^2 = 1$. Expand and rearrange in terms of t.

Open Kattis - goofy

`https://open.kattis.com/problems/wrapping`

Basic approach:

- Consider the line L passing through the given point p and point y (a parameter) on the vertical axis.
- Use the quadratic expression $||(1-t)\cdot(0,y)+t\cdot p||^2 = 1$. Expand and rearrange in terms of t.
- The line L is tangent to the circle if the discriminant Δ.

Open Kattis - goofy

`https://open.kattis.com/problems/wrapping`

Basic approach:

- Consider the line L passing through the given point p and point y (a parameter) on the vertical axis.
- Use the quadratic expression $||(1-t)\cdot(0,y)+t\cdot p||^2=1$. Expand and rearrange in terms of t.
- The line L is tangent to the circle if the discriminant Δ.
- The discriminant Δ is a quadratic in y, solve for $\Delta=0$ using the quadratic formula to get the two values of y.

Open Kattis - goofy

`https://open.kattis.com/problems/wrapping`

Basic approach:

- Consider the line L passing through the given point p and point y (a parameter) on the vertical axis.
- Use the quadratic expression $||(1-t)\cdot(0,y)+t\cdot p||^2=1$. Expand and rearrange in terms of t.
- The line L is tangent to the circle if the discriminant Δ.
- The discriminant Δ is a quadratic in y, solve for $\Delta=0$ using the quadratic formula to get the two values of y.

Caution: carefully check each step of your calculations!