Lebanese University
Faculty of Science
Computer Science BS Degree

Advanced Algorithms I3341

Dr Siba Haidar & Dr Antoun Yaacoub

Course Chapters

as per the textbook

1. Introduction
2. Data Structures and Libraries
3. Problem Solving Paradigms
4. Graph
5. Mathematics
6. String Processing
7. Computational Geometry

Computational Geometry

Chapter 7

Dr Siba Hadar & Dr Antoun Yaacoub

Typical issues with geometry problems

- ++ tricky corner test cases
  - vertical line → infinite gradient
  - collinear points
  - concave polygon
  - case where convex hull of a set of points is the set of points itself
- → do lots of corner test cases
- floating point precision errors
- tedious coding

Chapter Outline

1. Geometry Basics
   a. Points, Lines
   b. Circles
   c. Triangles
   d. Polygons
2. Algorithms on 2D polygons
   a. convex polygon
   b. concave polygon
   c. point inside | outside polygon
   d. cut polygon with straight line
   e. find convex hull of set of points

Not discussed

1. ~~Quadrilaterals~~
2. ~~3D Objects: Spheres~~
3. ~~Other 3D Objects: Cones, Cylinders, etc.~~
4. ~~Plane Sweep technique~~
5. ~~Intersection problems~~
6. ~~Divide and Conquer in geometry problems~~

Some Comp Geometry Principles

- highlight special cases
  - choose implementation to reduce them
- prefer test (predicates) over computing exact numerical answers
- floating point operations
  - avoid division, square root, …
  - operate in integers
  - floating point equality test: fabs(a - b) < EPS where EPS is a small number like 1e-9 instead of a == b

Geometry Basics

Point Simplest Representation

- point_i → integer
- basic raw form, minimalist mode
- whenever possible, work with point_i

```
class point_i
{
    int x, y;
        point_i() {
                x = y = 0;
    }

        point_i(int _x, int _y) {
                x = _x;
                y = _y;
    }
}
```

Point + Sorting Feature

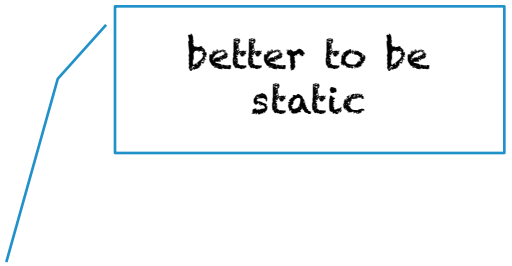
- only used if more precision is needed
- compareTo
  - use EPS (1e-9) to test float equal
  - useful for sorting
    - first: by x-coordinate
    - second: by y-coordinate

```
class point implements Comparable<point> {
   double x, y;

   point() {
      x = y = 0.0;
   }
   point(double _x, double _y) {
      x = _x;
      y = _y;
   }

   @Override
   public int compareTo(point other) {
      if (Math.abs(x - other.x) > EPS)
         return (int)
           Math.ceil(x - other.x);
      if (Math.abs(y - other.y) > EPS)
         return (int)
            Math.ceil(y - other.y);

      return 0;
   }

}
```


Dr Siba Hadar & Dr Antoun Yaacoub

Euclidean Distance between two points

- Math.hypot(dx, dy)
  - returns sqrt(dx * dx + dy * dy)
  - return double

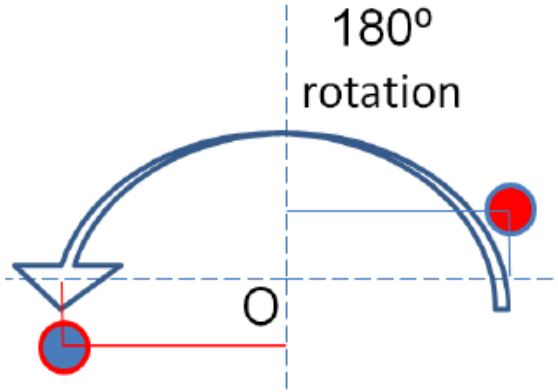



```
static double dist(point p1, point p2)
{
   return Math.hypot(p1.x - p2.x, p1.y - p2.y);
}
```

Rotate a point

- rotate a point
  - by angle θ
- counter clockwise (CCW)
- around O
  - using a rotation matrix:

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



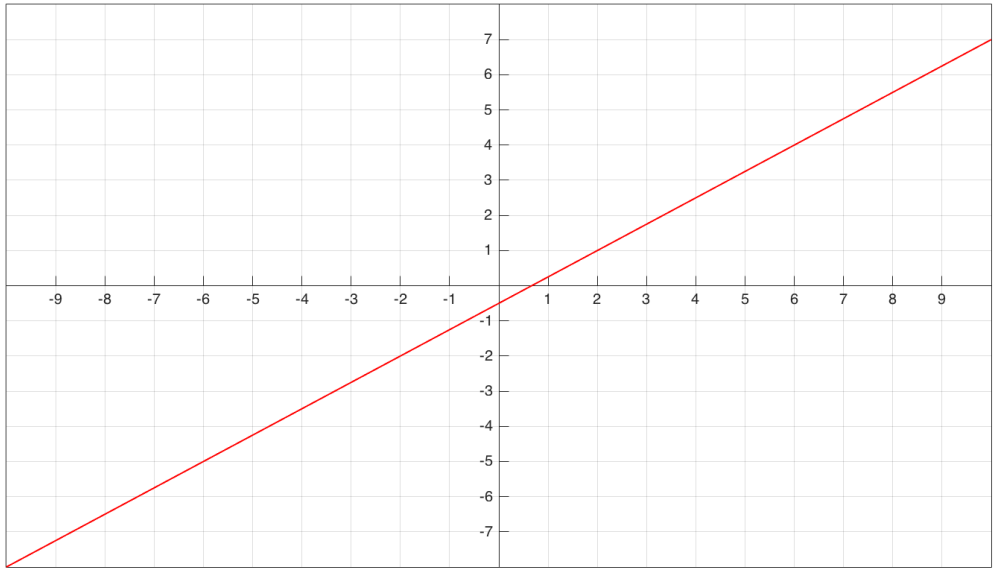

```
double DEG_to_RAD(double d){
  return d * Math.PI / 180.0;
}

double RAD_to_DEG(double r){
  return r * 180.0 / Math.PI;
}

point rotate(point p, double theta){
  double rad = DEG_to_RAD(theta);
  return new point(p.x * Math.cos(rad)
                       - p.y * Math.sin(rad),
                   p.x * Math.sin(rad)
                    + p.y * Math.cos(rad));
}
```

Lines

ch7_01_points_lines.cpp/java

Equation of Straight Line

- any straight line can be written in the form
  - $ax + by + c = 0$
- example
  - $y = \frac{3}{4}x - \frac{1}{2}$
  - can be written as
  - $4y - 3x + 2 = 0$
- in particular
  - equations $x = c$
  - cannot be as form $y = mx + c$



Lines

- poor line equation
  - y = mx + c
  - vertical line → special case


```
class line2 {
      double m, c;
}
```

- better line equation
  - ax + by + c = 0


```
class line {
      double a, b, c;
}
```

pointsToLine

- calculate a, b, c
  - → fill fields of 3rd parameter
- p1(x_1, y_1)
  - $ax_1 + by_1 + c = 0$ (1)
- p2(x_2, y_2)
  - $ax_2 + by_2 + c = 0$ (2)

- (1)-(2)→
  - $a(x_1 - x_2) + b(y_1 - y_2) = 0$
  - if $\partial x = x_1 - x_2 = \varepsilon$
    - vertical line $b = 0, a = 1, c = -x_1$
  - else
    - fix $b = 1.0$
    - $a(x_1 - x_2) + (y_1 - y_2) = 0$
    - $a = -\frac{\partial y}{\partial x}$ and $c = -ax_1 - y_1$

```
void pointsToLine ( point p1, point p2, line l)
{
   if (Math.abs(p1.x - p2.x) < EPS) {
      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;
      l.c = -(double) (l.a * p1.x) - p1.y;
   }
}
```

Interaction between two lines

- areParallel
  - check coefficients a & b

```
boolean areParallel(line l1, line l2) {
  return (Math.abs(l1.a-l2.a)<EPS)
          &&(Math.abs(l1.b-l2.b)<EPS);
}
```

- areSame
  - ++ check coefficient c

```
boolean areSame(line l1, line l2) {
  return areParallel(l1, l2)
          &&(Math.abs(l1.c-l2.c)<EPS);
}
```

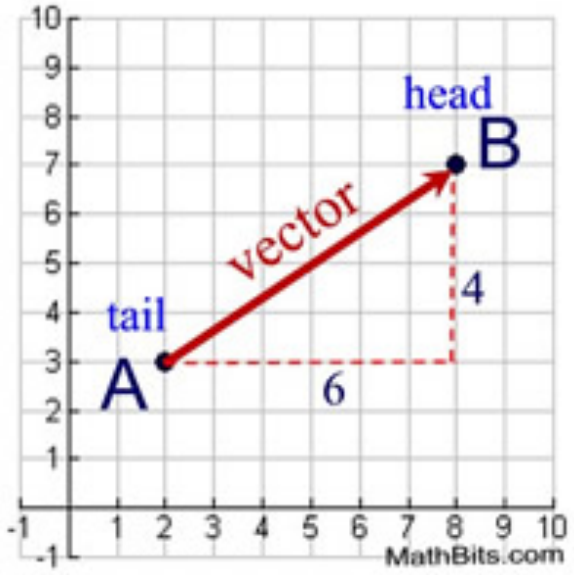
areIntersect

- if parallel → no
- else solve system of 2 linear algebraic equations with 2 unknowns
  - line1: $a_1x + b_1y + c_1 = 0$ (1)
  - line2: $a_2x + b_2y + c_2 = 0$ (2)
- x
  - $b_1 \times (2) - b_2 \times (1) = 0$
  - $x = \frac{b_2c_1 - b_1c_2}{a_2b_1 - a_1b_2}$
- y
  - b either 0 (vertival) or 1
  - avoid division by zero
  - →replace x value in non vertical equation

```
boolean areIntersect(line l1, line l2, point p) {
   if (areParallel(l1, l2))
     return false;
   p.x = (l2.b * l1.c - l1.b * l2.c)
         / (l2.a * l1.b - l1.a * l2.b);

   if (Math.abs(l1.b) > EPS)
      p.y = -(l1.a * p.x + l1.c);
   else
      p.y = -(l2.a * p.x + l2.c);
   return true;
}
```

Segment and Vector

- segment
  - line with two endpoints
  - finite length
- vector
  - segment with direction
- toVec
  - convert 2 points to vector




```
class vec {
  double x, y;
  vec(double _x, double _y) {
    x = _x;
    y = _y;
  }
}

vec toVec(point a, point b) {
  return new vec(b.x - a.x, b.y - a.y);
}
```

Scale and Translate

- scale a vector:
  - nonnegative s = [<1 .. 1 .. >1]
  - shorter .. same .. longer

```
vec scale(vec v, double s) {
  return new vec(v.x * s, v.y * s);
}
```

- translate: move point % vec

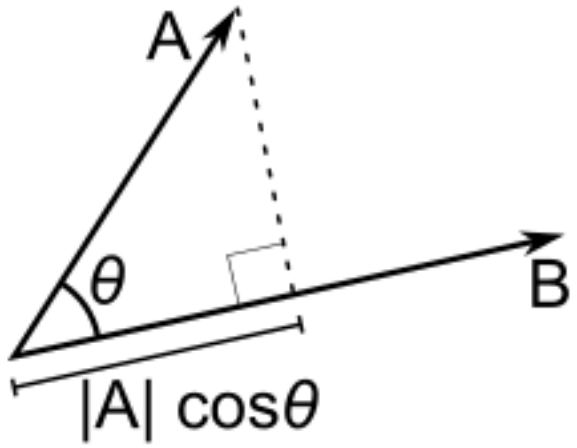
```
point translate(point p, vec v) {
  return new point(p.x + v.x, p.y + v.y);
}
```


Dr Siba Hadar & Dr Antoun Yaacoub

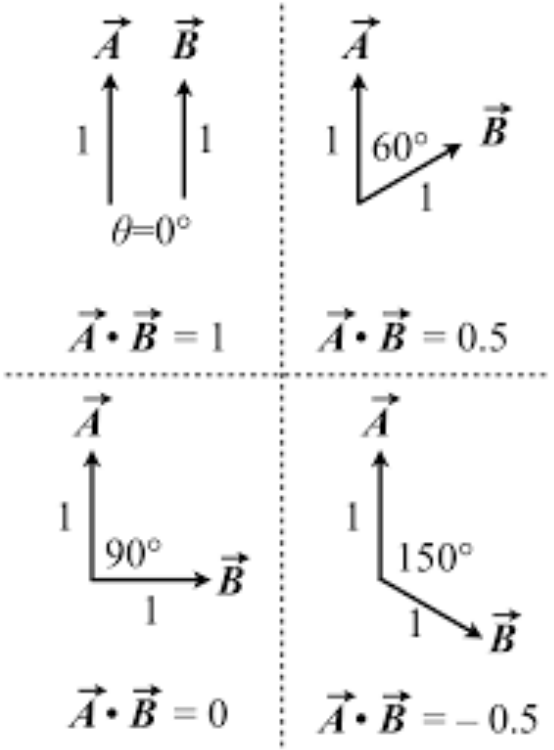
Dot Product

- dot product of 2 vectors
  - u=(a,b) and v=(c,d)
  - is denoted $u \cdot v$ read u dot v

- $\vec{u} \cdot \vec{v} = ac + bd$
  - if >0 → angle acute
  - if =0 → right angle
  - if <0 angle obtuse

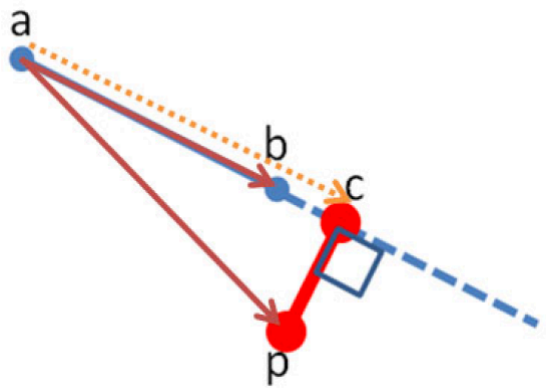



```
double dot(vec a, vec b) {
  return (a.x * b.x + a.y * b.y);
}
```



Distance to Line

- distance from p to line defined by a and b (distinct)
  - closest point stored in 4th param
  - formula: c = a + u * ab
  - translate a to c
  - dist(p,c)

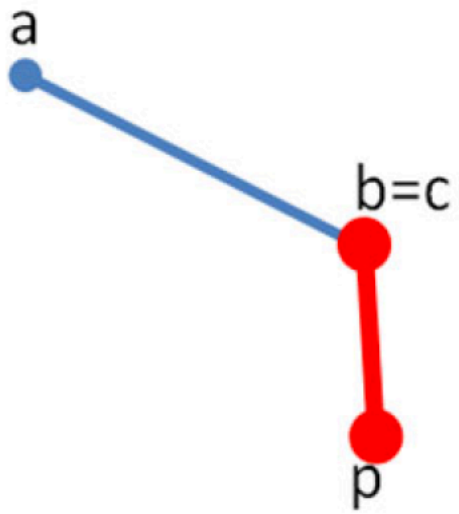



```
double norm_sq(vec v) {
  return v.x * v.x + v.y * v.y;
}
```

```
double distToLine(point p, point a, point b,
                  point c) {
  vec ap = toVec(a, p), ab = toVec(a, b);
  double u = dot(ap, ab) / norm_sq(ab);
  c = translate(a, scale(ab, u));
  return dist(p, c);
}
```

Distance to Line Segment

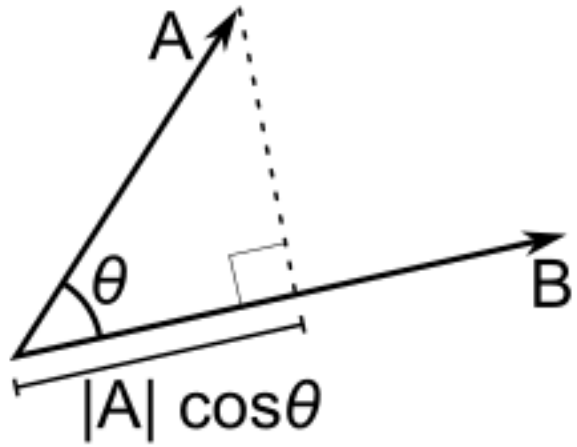
- dist from p to segment ab
- OK if a == b
- closest point stored in 4th param




```
double distToLineSegment(point p, point a,
       point b, point c) {
  vec ap = toVec(a, p),
      ab = toVec(a, b);
  double u = dot(ap, ab) / norm_sq(ab);

  if (u < 0.0) // closer to a
  {
    c = new point(a.x, a.y);
    return dist(p, a);
  }

  if (u > 1.0) // closer to b
  {
    c = new point(b.x, b.y);
    return dist(p, b);
  }

  return distToLine(p, a, b, c);
}
```

Angle

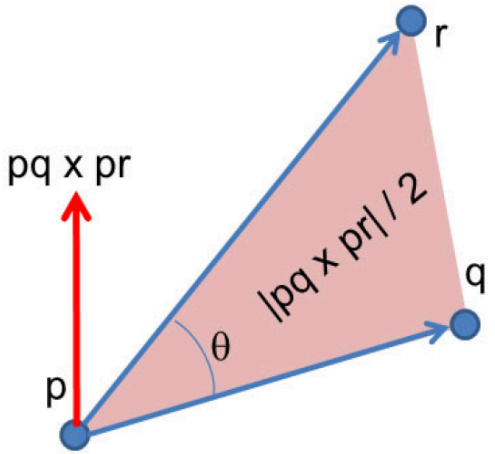
- angle in radiant aob using dot product
  - $oa \cdot ob = |oa||ob|cos(\theta)$
  - $theta = arcos(oa \cdot ob/|oa||ob|)$




```
double angle(point a, point o, point b) {
  vec oa = toVec(o, a),
      ob = toVec(o, b);

  return Math.acos(dot(oa, ob)
         / Math.sqrt(norm_sq(oa)
           * norm_sq(ob)));
}
```

Cross Product, CCW and Collinear Points

- line p q + point r
  - r on left | right side of the line?
  - 3 points p, q, r collinear?
- pq & pr 2 vectors
  - cross product $pq \times pr$ → vector ⊥
  - magnitude = area parallelogram span
    - > 0: p → q → r is a left turn
    - = 0: p, q , r are collinear
    - < 0: p → q → r is a right turn




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

boolean ccw(point p, point q, point r) {
  return cross(toVec(p, q),
               toVec(p, r)) > 0;
}

boolean collinear(point p, point q, point r) {
  return Math.abs(cross(toVec(p, q),
                        toVec(p, r))) < EPS;
}
```

Circles

ch7_02_circles.cpp/java

Circle

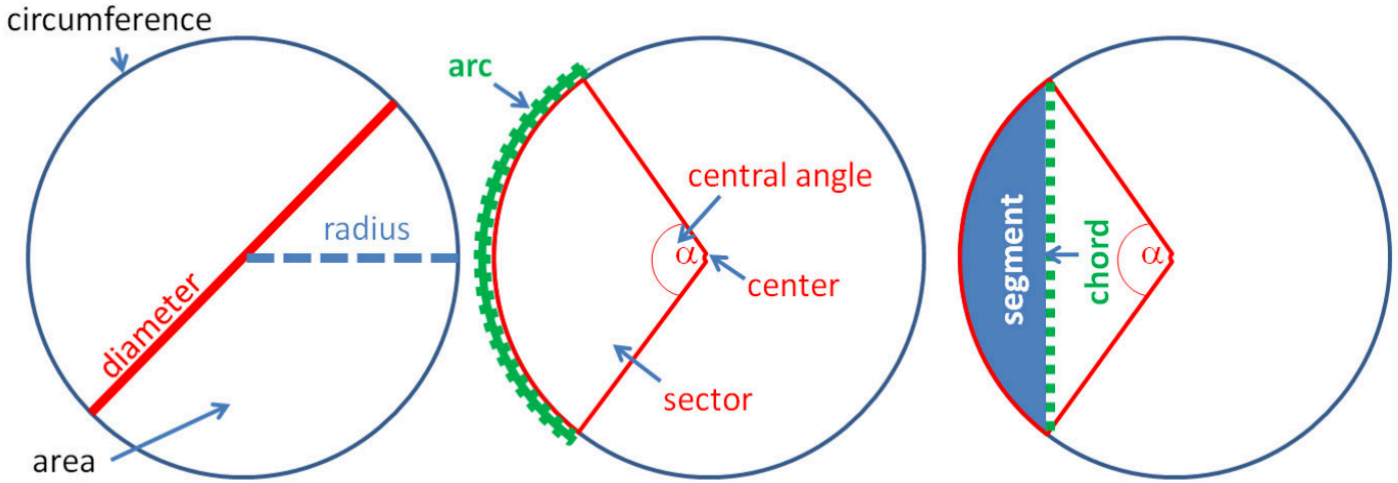
- circle
  - centered at (a, b)
  - radius r
  - set of points (x, y) /
  - $(x-a)^2+(y-b)^2=r^2$
- inCircle
  - 0 – inside
  - 1 – at border
  - 2 – outside
- modify for float

```
int insideCircle(point_i p, point_i c, int r) {
  int dx = p.x - c.x,
      dy = p.y - c.y;
  int Euc = dx * dx + dy * dy,
      rSq = r * r;

  return Euc<rSq? 0:Euc == rSq? 1:2;
}
```

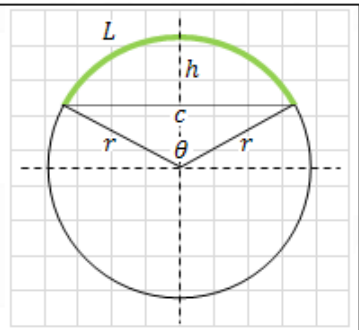
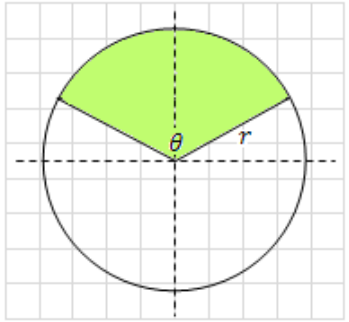
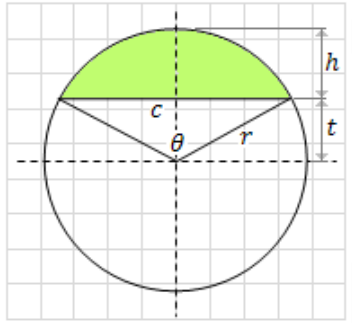
Circle

- for precision use
  - π = 2 * acos(0.0)
- diameter $d = 2r$
- circumference $c = \pi d$
- area $A = \pi r^2$

- $arc = \frac{\alpha}{360.0} \times c$
- $chord = \sqrt{2r^2(1 - cos(\alpha))}$
- $sector = \frac{\alpha}{360.0} \times A$
- segment area = sector –area of isosceles triangle with sides: r, r, chord



Summary

	$L = Arc\ length$ $L(r,\theta) = r\theta$ $L(r,c) = 2r\sin^{-1}\left(\frac{c}{2r}\right)$ $L(r,h) = 2r\cos^{-1}\left(1-\frac{h}{r}\right)$	$L = r\frac{\theta}{180}\pi$
Circle sector 	(θ in radians) $A(r,\theta) = \frac{\theta}{2}r^2$ $A(r,L) = \frac{1}{2}Lr$ $A(r,c) = r^2\sin^{-1}\left(\frac{c}{2r}\right)$	(θ in degree) $A = \frac{\theta}{360}\pi r^2$
	$P(r,\theta) = r(\theta+2)$ $P(r,c) = 2r\left(\sin^{-1}\left(\frac{c}{2r}\right)+1\right)$	$P = r\left(\frac{\theta}{180}\pi+2\right)$
Circle segment 	(θ in radians) $A(r,\theta) = \frac{r^2}{2}(\theta-\sin\theta)$ $A(r,c) = r^2\sin^{-1}\left(\frac{c}{2r}\right)-\frac{c}{4}\sqrt{4r^2-c^2}$ $A(r,t) = r^2\cos^{-1}\left(\frac{t}{r}\right)-t\sqrt{r^2-t^2}$ $A(r,h) = r^2\cos^{-1}\left(1-\frac{h}{r}\right)-(r-h)\sqrt{2rh-h^2}$	(θ in degrees) $A = \frac{r^2}{2}\left(\frac{\pi}{180}\theta-\sin\theta\right)$
	$P(r,\theta) = r\left(\theta+2\sin\frac{\theta}{2}\right)$ $P(r,c) = r^2\sin^{-1}\left(\frac{c}{2r}\right)-\frac{c}{2}\sqrt{4r^2-c^2}$	$P = r\left(\frac{\pi}{180}\theta+2\sin\frac{\theta}{2}\right)$

source http://www.ambrsoft.com/TrigoCalc/Sphere/Arc_.htm

Find the center

- p1 & p2 + radius r
  - → ? centers (c1 & c2)
- to get the other center, reverse p1 and p2

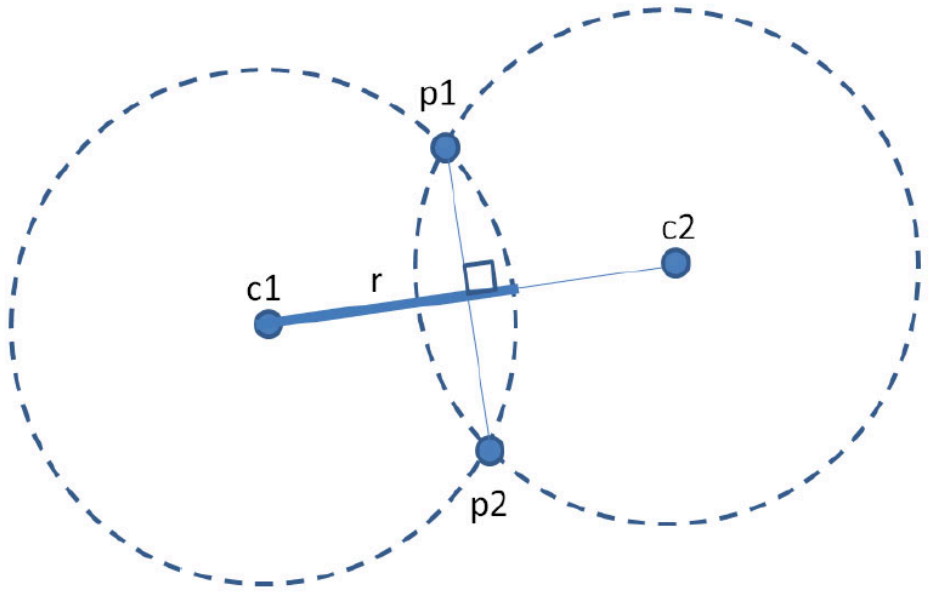


Figure 7.4: Circle Through 2 Points and Radius

```
boolean circle2PtsRad (point p1, point p2,
                                  double r, point c) {
  double d2 = (p1.x - p2.x)*(p1.x - p2.x)
            + (p1.y - p2.y)*(p1.y - p2.y);
  double det = r * r / d2 - 0.25;

  if (det < 0.0)
    return false;

  double h = Math.sqrt(det);

  c.x = (p1.x + p2.x) * 0.5
        + (p1.y - p2.y) * h;

  c.y = (p1.y + p2.y) * 0.5
        + (p2.x - p1.x) * h;

  return true;
}
```

Triangles

ch7_03_triangles.cpp/java

Triangles

- Polygon 3 vertices 3 edges
- Area of Triangle
  - A = 0.5 * base x height
- Perimeter p = a + b + c
  - a, b, c 3 edges

- Heron's formula
  - Area = sqrt(s*(s-a)*(s-b)*(s-c))
  - s = 0.5 * p : semi-perimeter
  - safer from overflow:
    - A = sqrt(s)*sqrt(s-a)*sqrt(s-b)*sqrt(s-c)
    - slightly more imprecise

```
double perimeter (double ab, double bc, double
ca) {
  return ab + bc + ca;
}

double perimeter (point a, point b, point c){
  return dist(a, b)
         + dist(b, c) + dist(c, a);
}

double area (double ab, double bc, double ca) {
  double s = 0.5 * perimeter(ab, bc, ca);
  return Math.sqrt(s)
         * Math.sqrt(s - ab)
         * Math.sqrt(s - bc)
         * Math.sqrt(s - ca);
}

double area (point a, point b, point c) {
  return area(dist(a, b),
              dist(b, c), dist(c, a));
}
```

Types of triangles

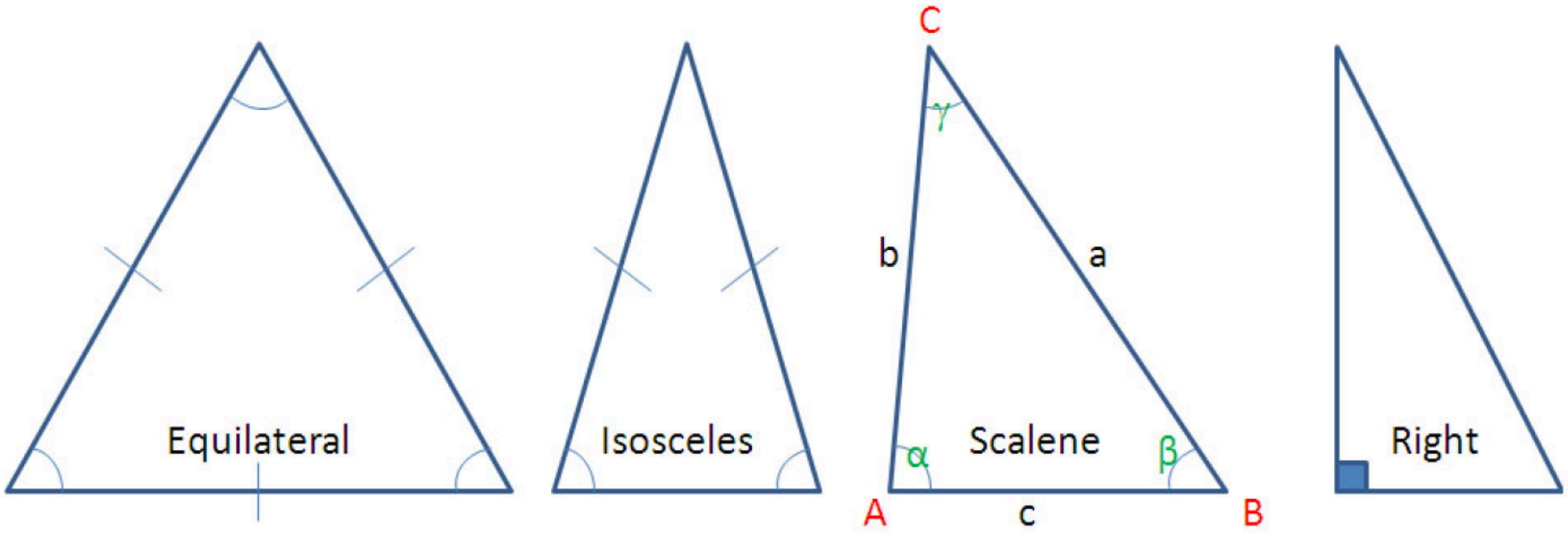
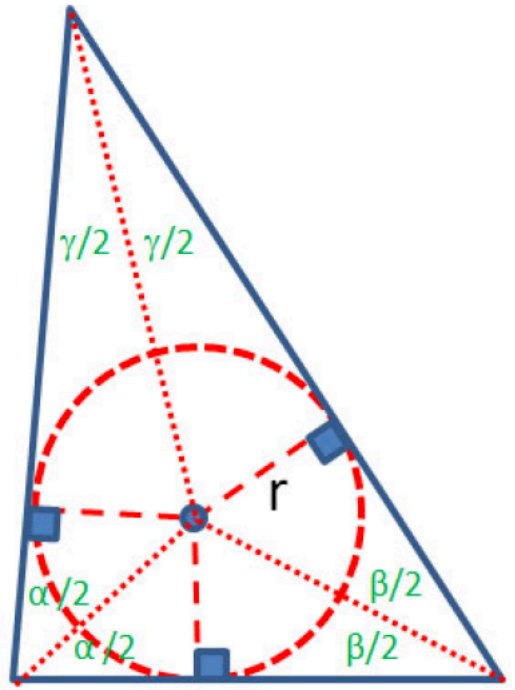


Figure 7.5: Triangles

InCircle

- 3 points p, q, r
- ? circumcenter c1 & radius R1
- of inner / inscribed circle / incircle



InCircle

- 3 points p, q, r
- ? circumcenter c1 & radius R1
- of inner / inscribed circle / incircle

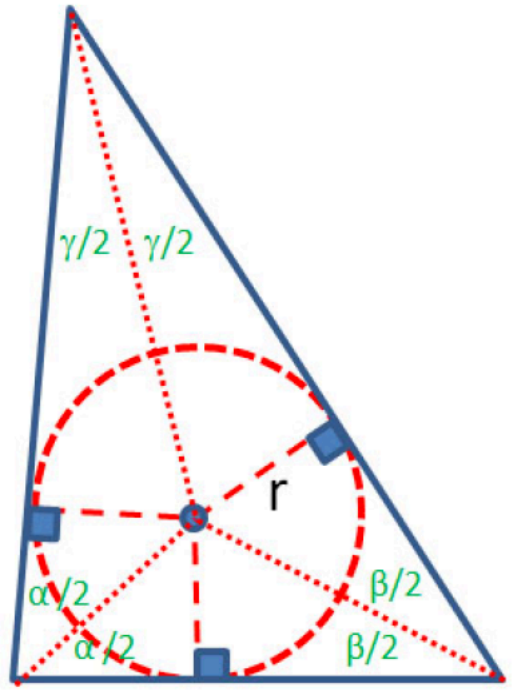
- triangle with area A and semi-perimeter s
  - inscribed circle (incircle) with
  - radius r = A/s

```
double rInCircle (double ab,
                  double bc, double ca) {
  return area(ab, bc, ca) / (0.5 *
perimeter(ab, bc, ca));
}

double rInCircle (point a, point b, point c) {
  return rInCircle( dist(a, b),
                    dist(b, c), dist(c,a));
}
```

InCircle

- 3 points p, q, r
- ? circumcenter c1 & radius R1
- of inner / inscribed circle / incircle

- center of incircle
  - meeting point between triangle's angle bisectors
  - 2 angle bisectors and find intersection



InCircle

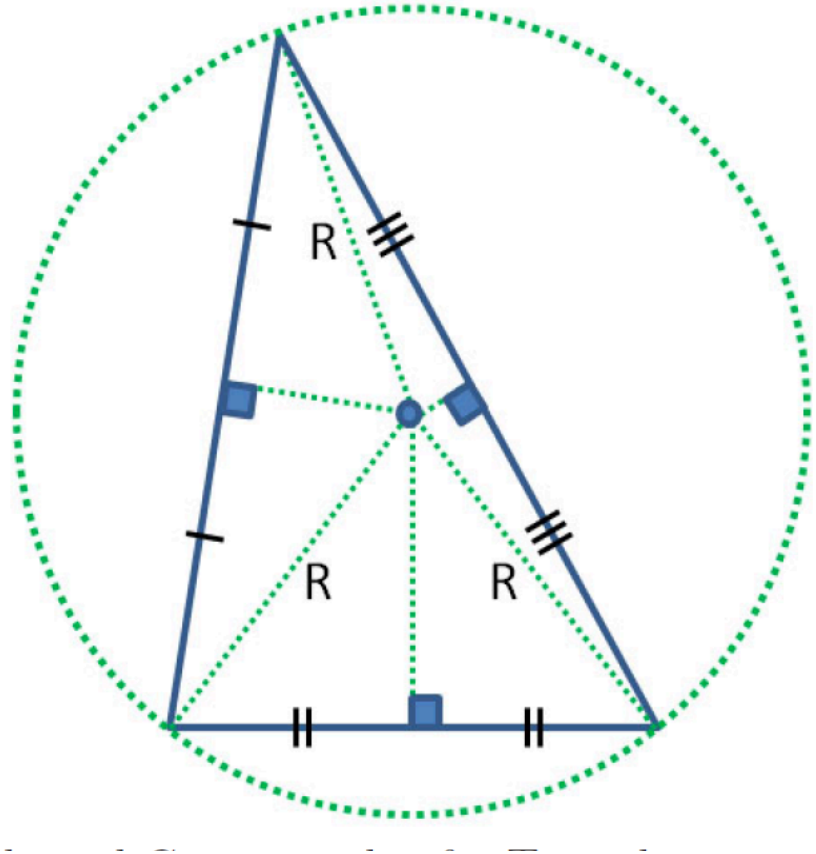
- 3 points p, q, r
- ? circumcenter c1 & radius R1
- of inner / inscribed circle / incircle

- returns
  - 1 if there is an inCircle
    - center ctr radius r
  - 0 otherwise

```
int inCircle (point p1, point p2, point p3,
                          point ctr, double r) {
  r = rInCircle(p1, p2, p3);
  if (Math.abs(r) < EPS)
    return 0;
  line l1 = new line(),
       l2 = new line();
  double ratio = dist(p1, p2) / dist(p1, p3);
  point p;
  p = translate(p2,
            scale(toVec(p2, p3),
                  ratio / (1 + ratio)));
  pointsToLine(p1, p, l1);
  ratio = dist(p2, p1) / dist(p2, p3);
  p = translate(p1,
            scale(toVec(p1, p3),
                  ratio / (1 + ratio)));
  pointsToLine(p2, p, l2);
  areIntersect(l1, l2, ctr);
  return 1;
}
```

CircumCircle

- 3 points p, q, r
- ? circumcenter c2 & radius R2
- of outer / circumscribed circle / circumcircle



CircumCircle

- 3 points p, q, r
- ? circumcenter c2 & radius R2
- of outer / circumscribed circle / circumcircle

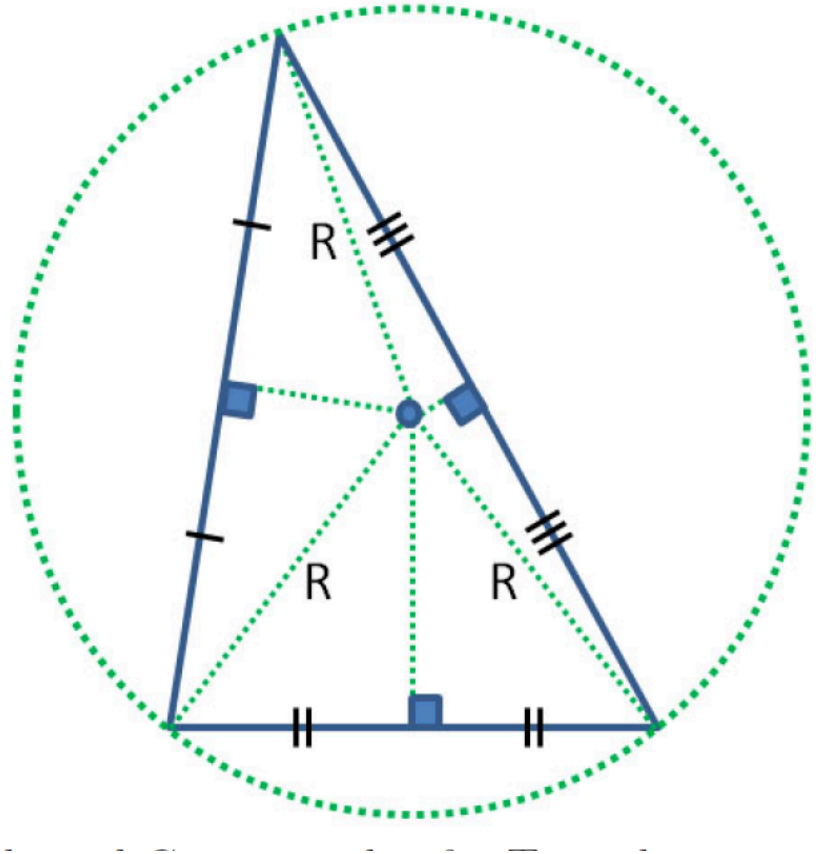
- 3 sides: a, b, c and area A
  - circumcircle radius
  - R = abc/(4A)

```
double rCircumCircle(double ab, double bc,
                     double ca) {
  return ab * bc * ca
        /(4.0 * area(ab, bc, ca));
}

double rCircumCircle(point a, point b,
                     point c) {
  return
    rCircumCircle(dist(a,b),
                  dist(b,c), dist(c,a));
}
```

CircumCircle

- 3 points p, q, r
- ? circumcenter c2 & radius R2
- of outer / circumscribed circle / circumcircle

- center = meeting point between triangle's perpendicular bisectors



CircumCircle

- 3 points p, q, r
- ? circumcenter c2 & radius R2
- of outer / circumscribed circle / circumcircle

```
double circumCircle(point p1, point p2,
                    point p3, point ctr) {
  double a = p2.x - p1.x,
         b = p2.y - p1.y;
  double c = p3.x - p1.x,
         d = p3.y - p1.y;
  double e = a * (p1.x + p2.x)
           + b * (p1.y + p2.y);
  double f = c * (p1.x + p3.x)
           + d * (p1.y + p3.y);
  double g = 2.0 * (a * (p3.y - p2.y)
           - b * (p3.x - p2.x));
  if (Math.abs(g) < EPS)
    return 0;

  ctr.x = (d * e - b * f) / g;
  ctr.y = (a * f - c * e) / g;

  return dist(p1, ctr);
}
```

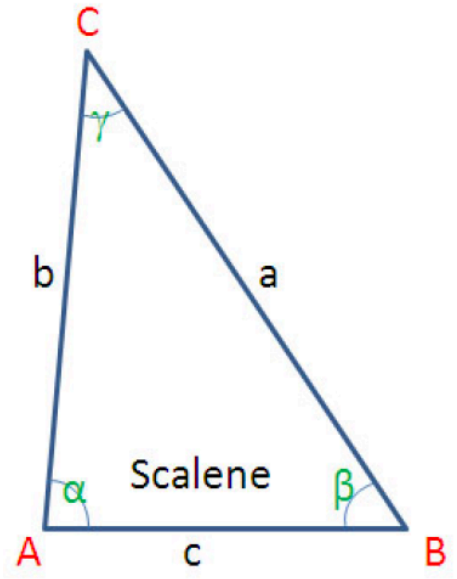
Triangle inequalities

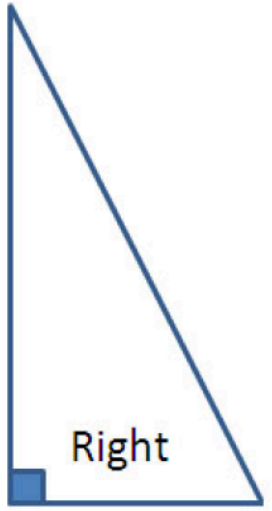
- To check if three line segments
  - length a <= b <= c
  - can form a triangle

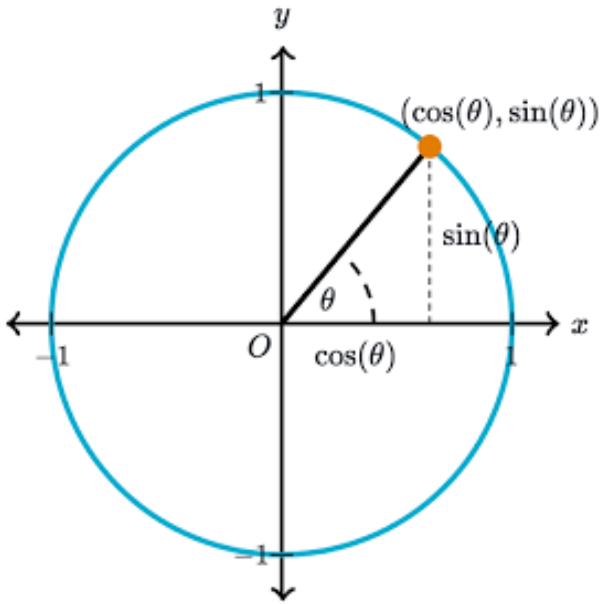
- check if a + b > c

Trigonometry

- Law of Cosines
  - $c^2 = a^2 + b^2 - 2ab\cos\gamma$
- Law of Sines
  - $\frac{a}{\sin\alpha} = \frac{b}{\sin\beta} = \frac{c}{\sin\gamma} = 2R$
  - R radius of circumcircle
- Phytagorean Theorem
  - right triangle
  - $c^2 = a^2 + b^2$
    - $\cos\frac{\pi}{2} = 0$
- Pythagorean Triple (a, b, c)
  - such that $c^2 = a^2 + b^2$
  - if (a, b, c) is a Pythagorean triple
  - then so is (ka, kb, kc)
  - for any positive integer k

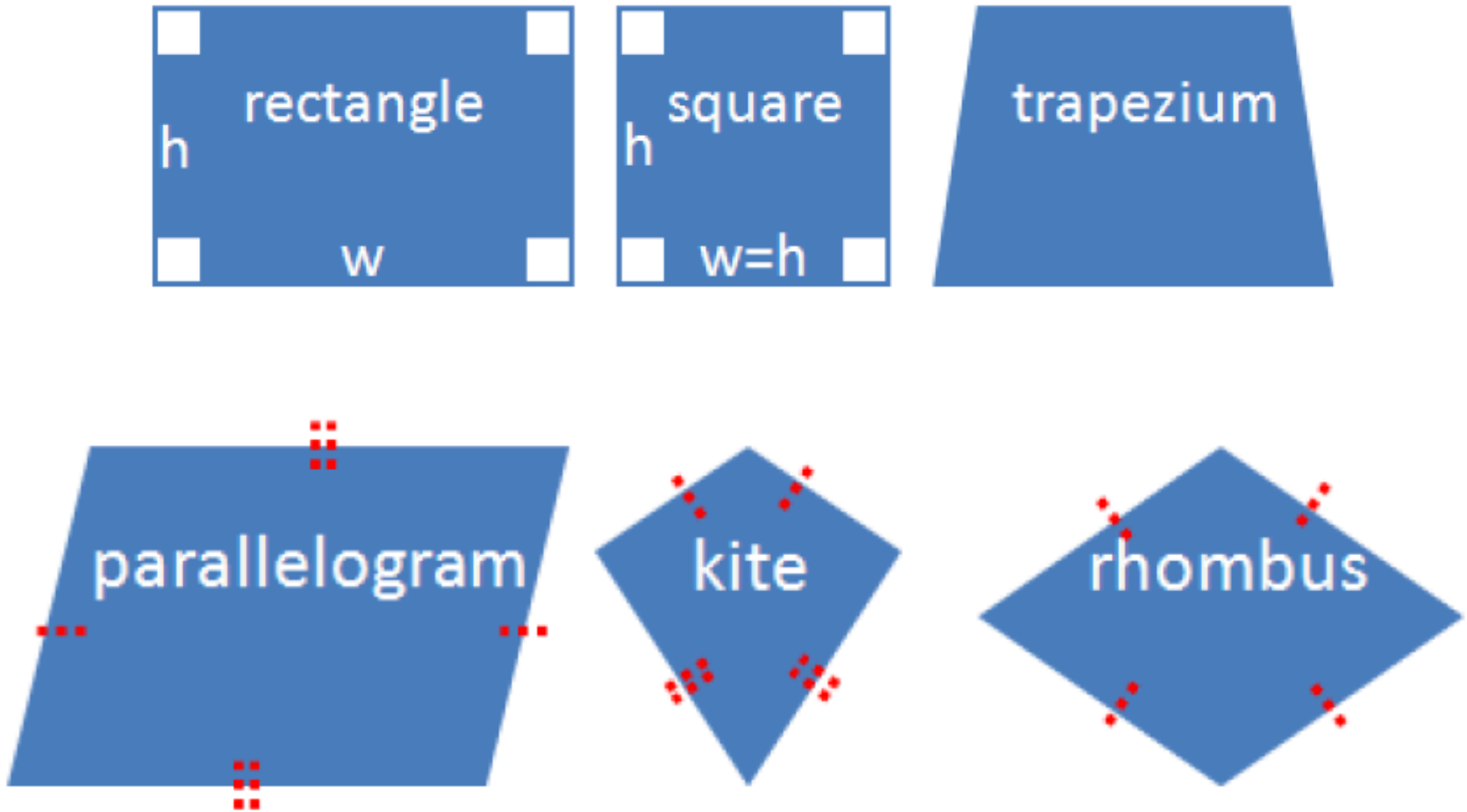






Dr Siba Hadar & Dr Antoun Yaacoub

Quadrilaterals

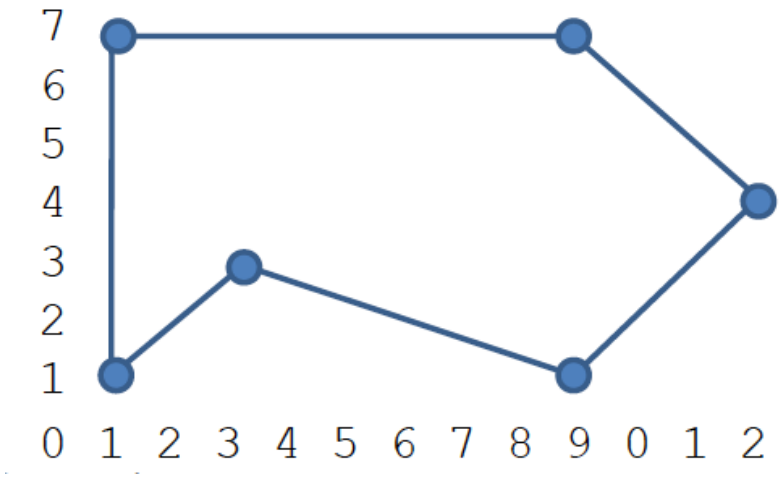


Algorithms on Polygon

ch7_04_polygon.cpp/java

Polygon

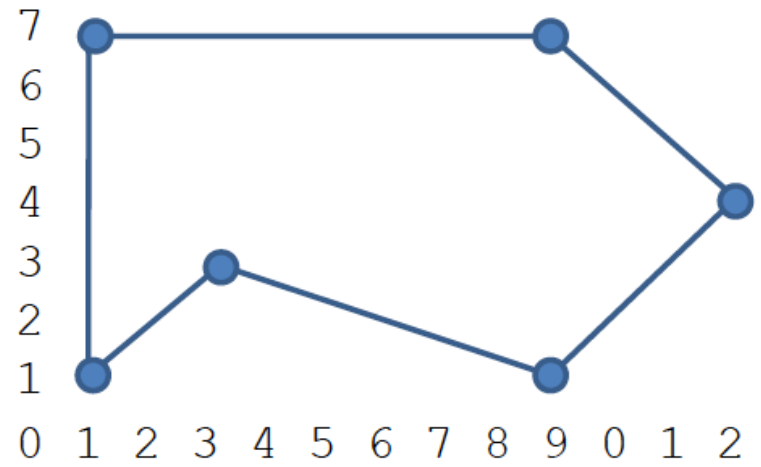
- plane figure bounded by
  - closed circuit
  - composed of finite sequence of straight line segments
- basic form
  - vertices ordered in cw or ccw order
- usually first = last vertex
- https://visualgo.net/en/polygon




```
List<point> P = new ArrayList<point>();
P.add(new point(1, 1));
P.add(new point(3, 3));
P.add(new point(9, 1));
P.add(new point(12, 4));
P.add(new point(9, 7));
P.add(new point(1, 7));
P.add(P.get(0)); // loop back
```

Perimeter of polygon (trivial)

- perimeter of polygon
  - convex or concave
  - with n vertices
  - given in CW or CCW
- computed via simple function




```
double perimeter(List<point> P) {

  double result = 0.0;
  for (int i = 0; i<(int)P.size()-1; i++)
    result += dist(P.get(i),P.get(i+1));

  return result;
}
```

Area of a Polygon

- simple polygon non-self-intersecting
  - $A = \frac{1}{2}\begin{bmatrix} x_1 & \cdots & y_1 \\ \vdots & \ddots & \vdots \\ x_n & \cdots & y_n \end{bmatrix}$
  - $= \frac{1}{2}\sum_{i=1}^{n}(x_i y_{i+1} - x_{i+1} y_i)$
  - beware with last point
- A
  - >0 vertices CCW
  - <0 vertices CW
- https://en.wikipedia.org/wiki/Polygon#Area

```
double area(List<point> P) {
  double result = 0.0, x1, y1, x2, y2;
  for (int i = 0;
       i < (int)P.size()-1; i++) {
    x1 = P.get(i).x;
    x2 = P.get(i+1).x;
    y1 = P.get(i).y;
    y2 = P.get(i+1).y;
    result += (x1 * y2 - x2 * y1);
  }
  return Math.abs(result) / 2.0;
}
```

Check if a Polygon is Convex

- convex polygon
  - → any line segment drawn inside the polygon does not intersect any edge of the polygon
  - otherwise Concave

- convex?
  - check whether all 3 consecutive vertices form same turns
    - all left turns ccw, or
    - all right turns cw
  - 1 counter example? → concave

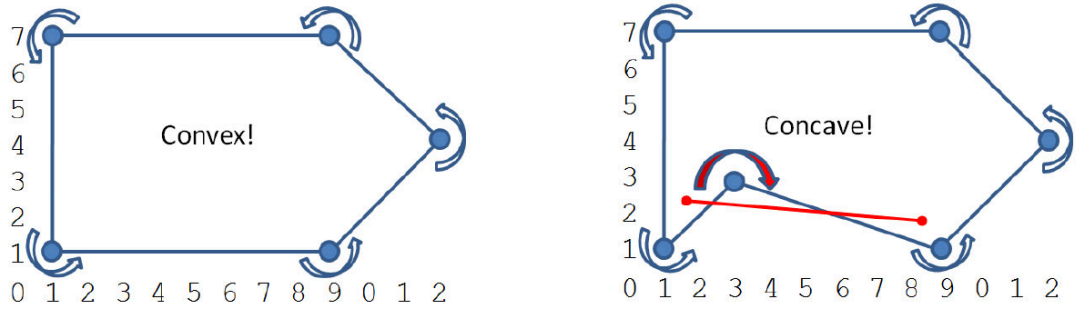
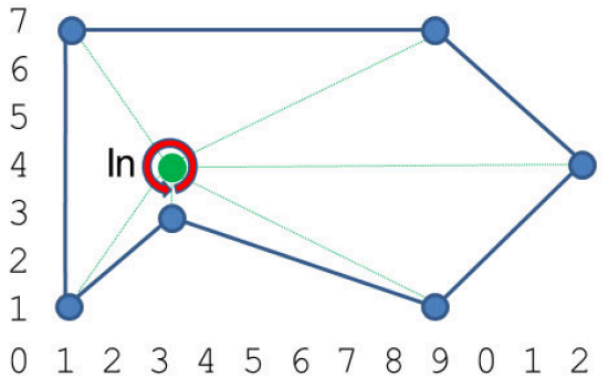


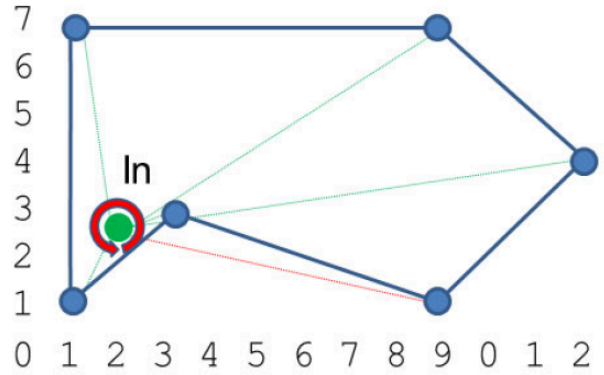
Figure 7.8: Left: Convex Polygon, Right: Concave Polygon

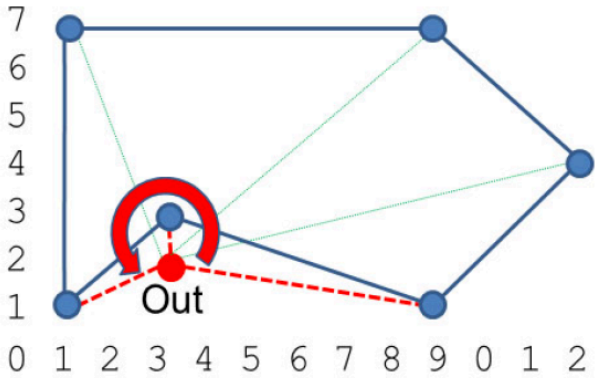
```
boolean isConvex(List<point> P) {
  int sz = (int)P.size();
  if (sz <= 3) return false;
  boolean isLeft =
    ccw(P.get(0), P.get(1), P.get(2));
  // remember one result
  for (int i = 1; i < sz-1; i++)
    if (ccw(P.get(i), P.get(i+1),
        P.get((i+2) == sz ? 1 : i+2))
      != isLeft)
      return false;
  return true;
}
```

Check if a Point is Inside a Polygon

- winding number algorithm
- check for either convex or concave polygon
- sum of angles $\Sigma p_i \widehat{ptp_{i+1}}$
  - CCW angle >0
  - CW angle <0
- if sum = 2π
  - pt is inside polygon P
- else→ outside







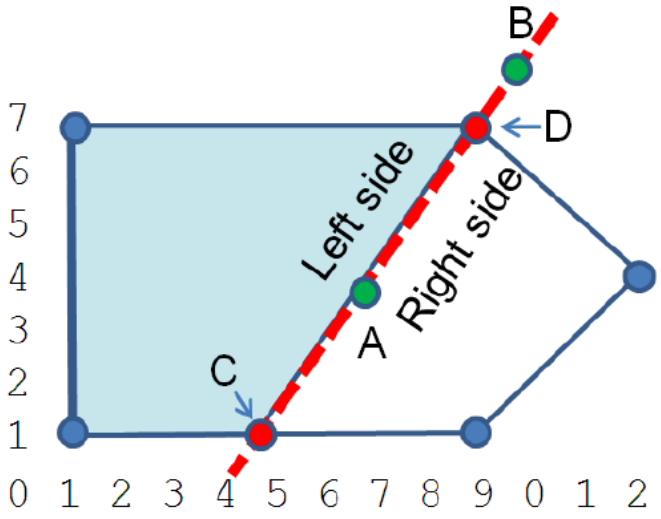
Check if a Point is Inside a Polygon

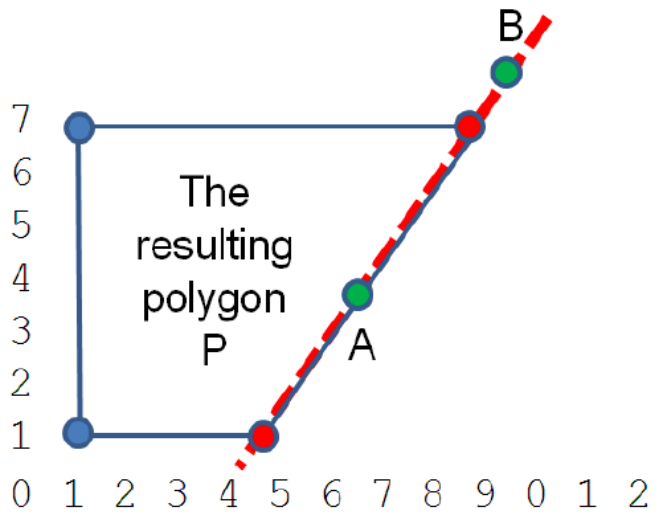
- winding number algorithm
- check for either convex or concave polygon
- sum of angles $\Sigma p_i \widehat{ptp_{i+1}}$
  - CCW angle >0
  - CW angle <0
- if sum = 2π
  - pt is inside polygon P
- else→ outside

```
boolean inPolygon(point pt, List<point> P) {
  if ((int)P.size() == 0)
    return false;
  double sum = 0;

  for (int i = 0; i < (int)P.size()-1; i++)
  {
    if (ccw(pt, P.get(i), P.get(i+1)))
      sum += angle(P.get(i), pt, P.get(i+1));
    else
      sum -= angle(P.get(i), pt, P.get(i+1));
  }

  return Math.abs(Math.abs(sum)-2*Math.PI)<EPS;

}
```

Cut a Polygon with a Straight Line

- line (AB) cut convex polygon Q
  - → 2 convex sub-polygons
- visit vertices v_i one by one
  - ABv_i CCW
    - → add v_i to P
  - $(AB) \cap [v_i v_{i+1}] = I \neq \emptyset$
    - → add I as new vertex in P if not a vertex already
  - ABv_i CW
    - → do nothing





Cut a Polygon with a Straight Line

- line (AB) cut convex polygon Q
  - → 2 convex sub-polygons
- visit vertices v_i one by one
  - ABv_i CCW
    - → add v_i to P
  - $(AB) \cap [v_i v_{i+1}] = I \neq \emptyset$
    - → add I as new vertex in P if not a vertex already
  - ABv_i CW
    - → do nothing
- can use areIntersect

```
point lineIntersectSeg(point p, point q,
                       point A, point B) {

  double a = B.y-A.y;
  double b = A.x-B.x;

  double c = B.x*A.y-A.x*B.y;

  double u = Math.abs(a*p.x+b*p.y+c);
  double v = Math.abs(a*q.x+b*q.y+c);

  return new point((p.x*v+q.x*u)/(u+v),
                   (p.y*v+q.y*u)/(u+v));
}
```

Cut a Polygon with a Straight Line

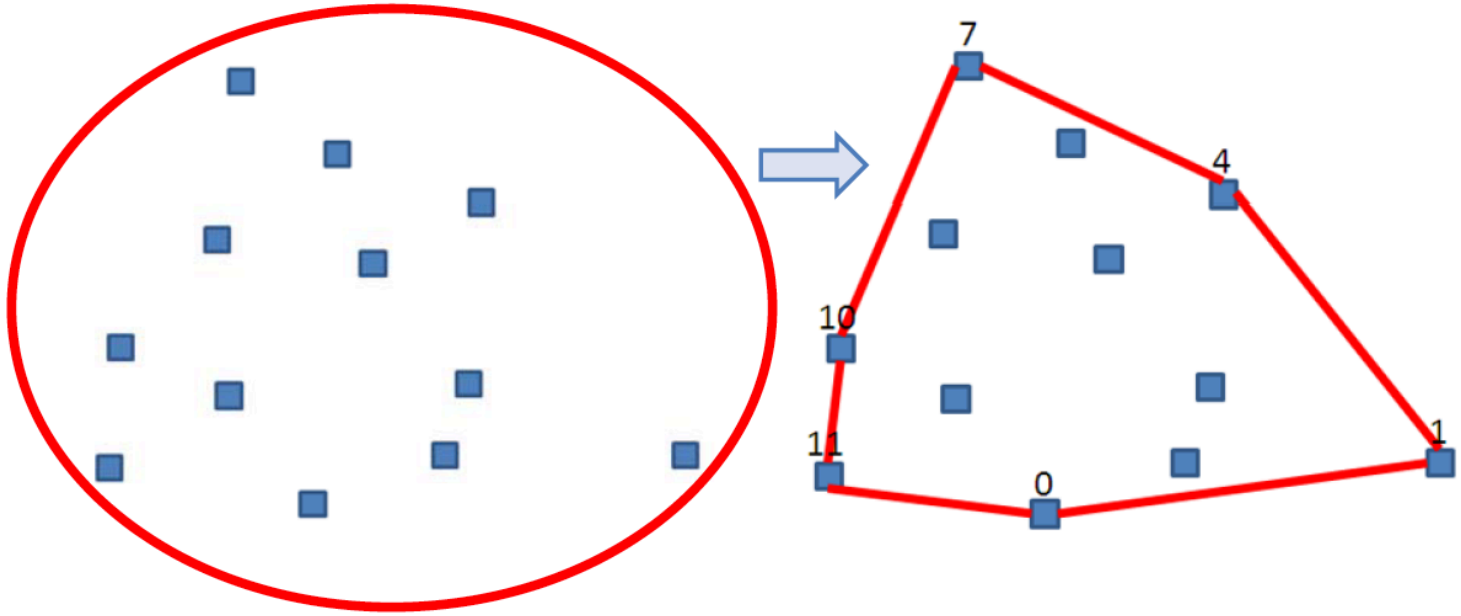
- line (AB) cut convex polygon Q
  - → 2 convex sub-polygons
- visit vertices v_i one by one
  - ABv_i CCW
    - → add v_i to P
  - $(AB) \cap [v_i v_{i+1}] = I \neq \emptyset$
    - → add I as new vertex in P if not a vertex already
  - ABv_i CW
    - → do nothing
- cross product slide

```
List<point> cutPolygon(point a, point b,
                       List<point> Q) {
  List<point> P = new ArrayList<point>();

  for (int i = 0; i < (int)Q.size(); i++) {
    double left1 = cross(toVec(a, b),
                         toVec(a, Q.get(i))),
           left2 = 0;

    if (i != (int)Q.size()-1)
      left2 = cross(toVec(a, b),
                    toVec(a, Q.get(i+1)));

    if (left1 > -EPS)
      P.add(Q.get(i));

    if (left1 * left2 < -EPS)
      P.add(lineIntersectSeg(Q.get(i),
                             Q.get(i+1),a,b));
  }

  if (!P.isEmpty()
      && P.get(P.size()-1).compareTo(P.get(0))!=0)
    P.add(P.get(0));

  return P;
}
```

Find Convex Hull of a Set of Points

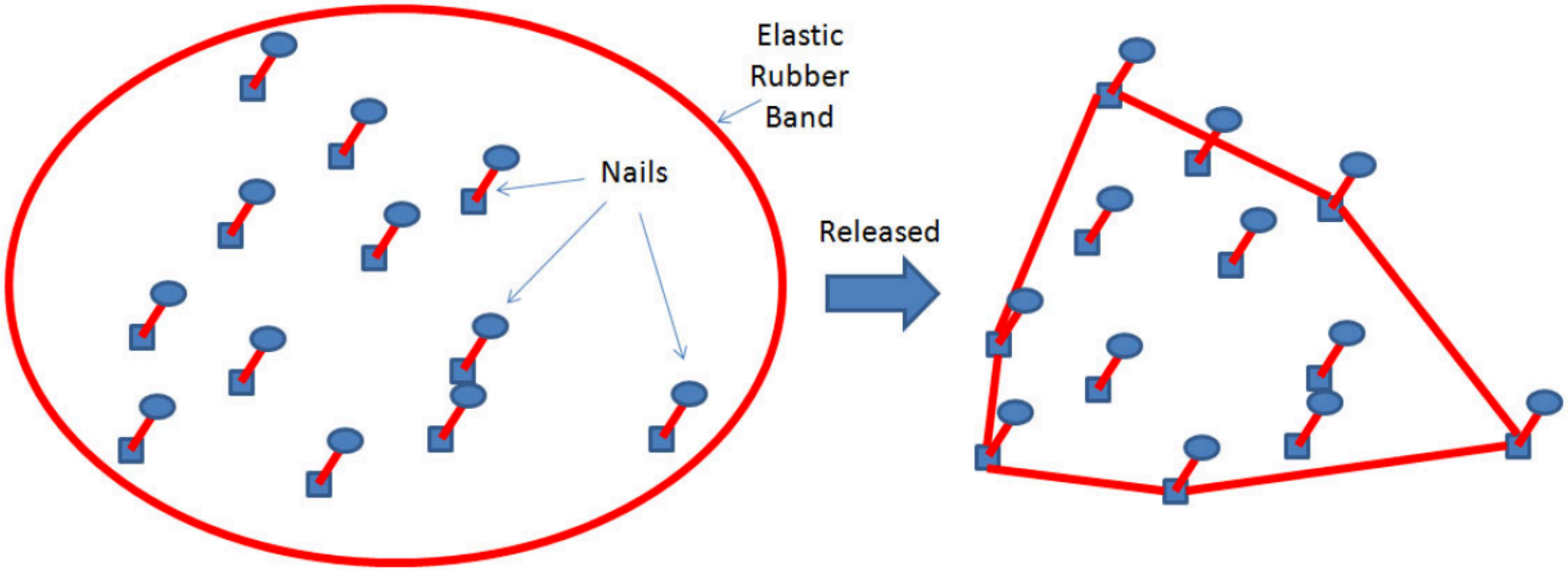
- The convex hull of set of points P
  - smallest convex polygon CH(P)
  - for which each point in P is
    - either on the boundary of CH(P) or
    - in its interior

- https://visualgo.net/en/convexhull



Find Convex Hull of a Set of Points

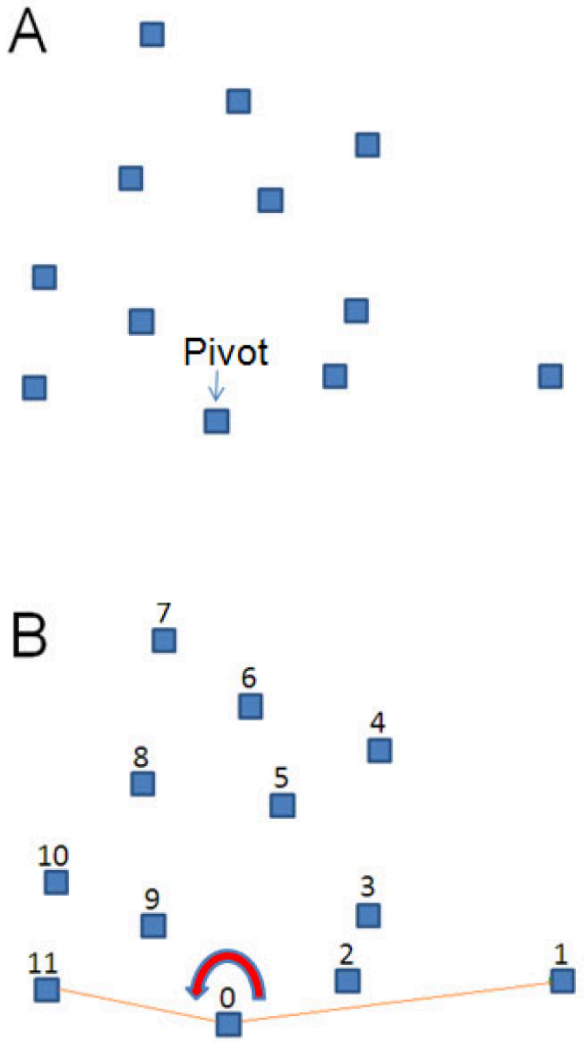
- if
  - points = nails
  - rubber band released → enclose as small an area as possible
- → convex hull



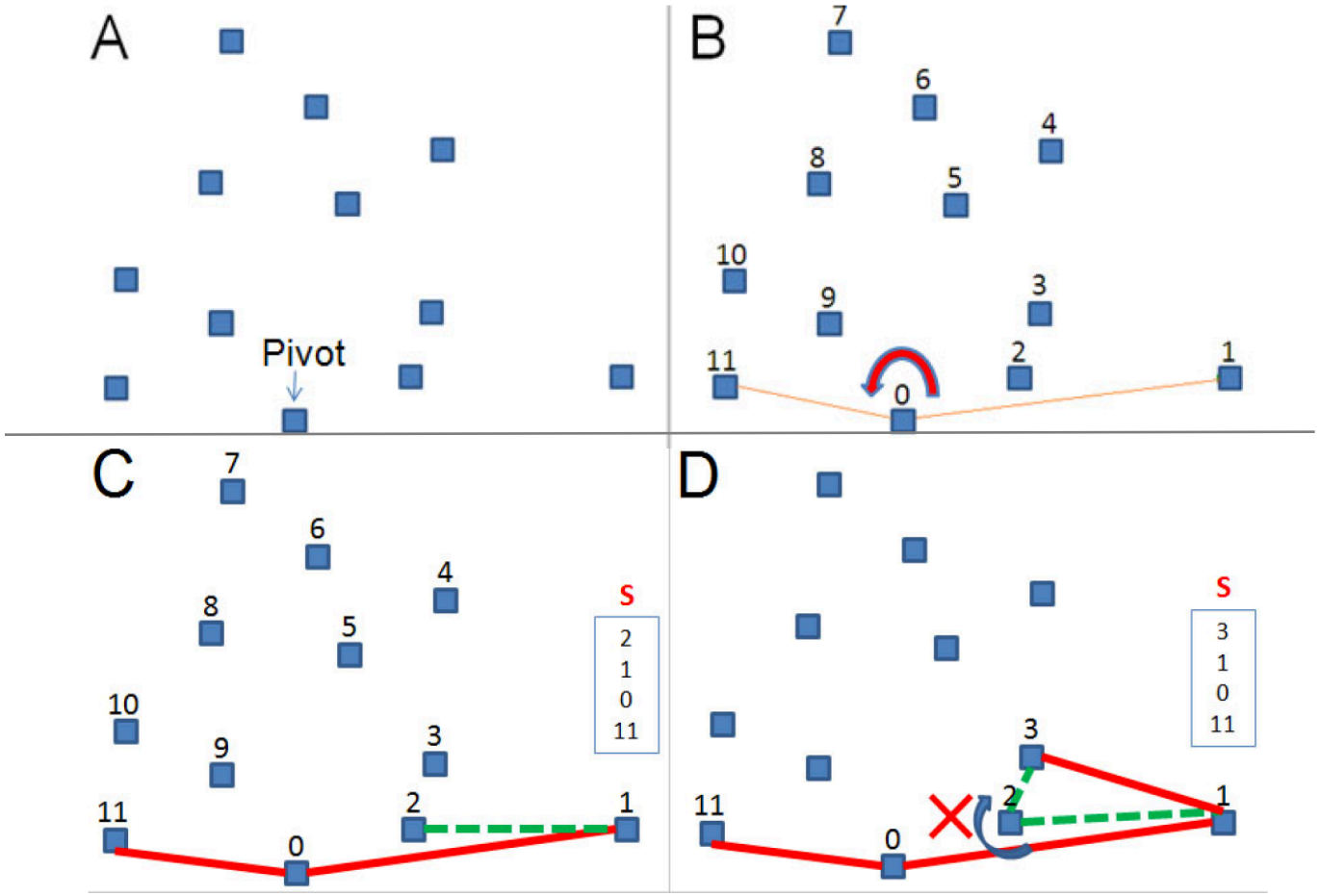
- Finding convex hull of a set of points has natural applications in packing problems.

Graham's Scan algorithm $O(n \log n)$

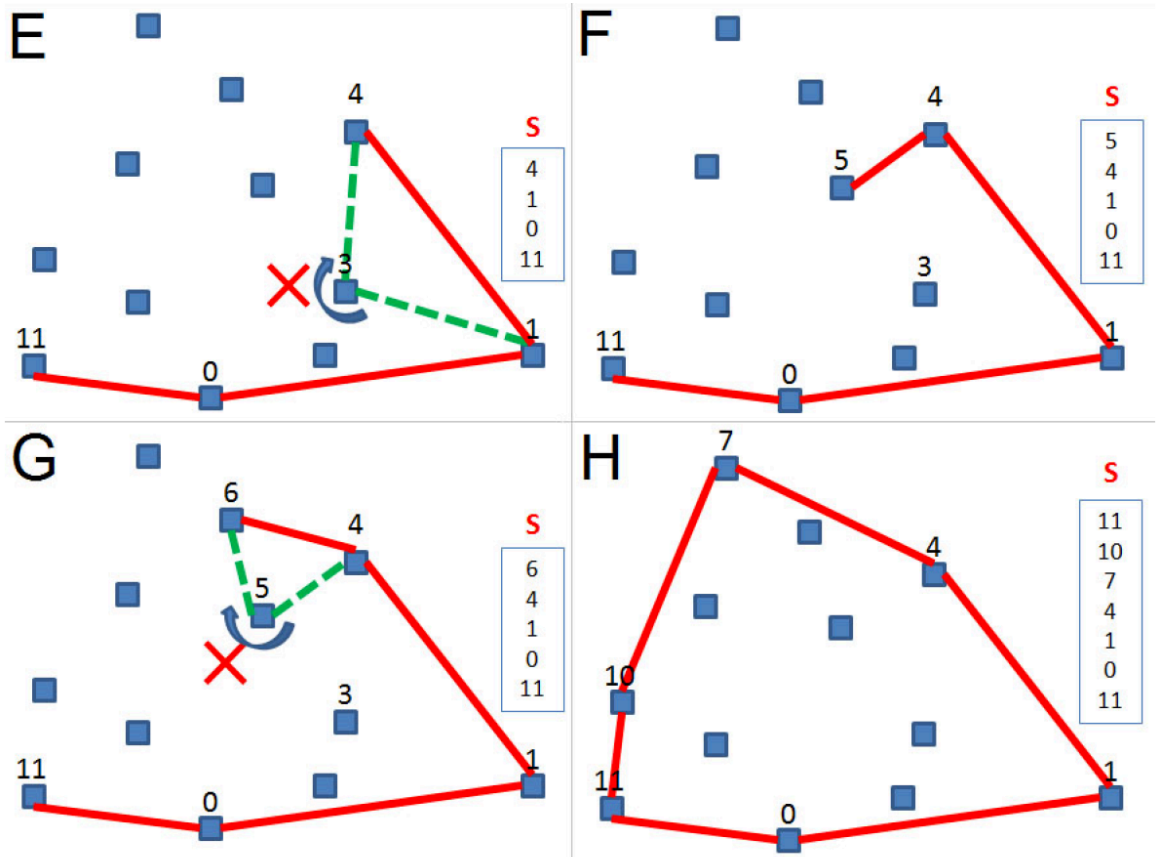
- pivot
  - bottom most, right most point
- angular sorting w.r.t pivot
  - easy with library
- series of ccw tests
  - stack S of candidate points
  - invariant
    - top 3 items in stack S must be CCW
  - initially
    - insert 3 points: N-1, 0, and 1
    - → always CCW
    - because 0 is the pivot!
  - then
    - each vertex v in sort
    - if top 2 with v are not going to be CCW
      - pop the last vertex added
    - else
      - push v int S



Graham's Scan algorithm



Graham's Scan algorithm



Graham's Scan algorithm

not the best programming practice

```
point pivot = new point();

List<point> CH(List<point> P) {
  int i, j, n = (int)P.size();
  if (n <= 3) {
    if (P.get(0).compareTo(P.get(n-1)) != 0)
      P.add(P.get(0));
    return P;
  }

  int P0 = 0;
  for (i = 1; i < n; i++)
    if (P.get(i).y  < P.get(P0).y
       ||(P.get(i).y == P.get(P0).y && P.get(i).x>P.get(P0).x))
      P0 = i;

  point temp = P.get(0);
  P.set(0, P.get(P0));
  P.set(P0 ,temp);
  pivot = P.get(0);
  Collections.sort(P, new AngComp());

  List<point> S = new ArrayList<point>();
  S.add(P.get(n-1));
  S.add(P.get(0));
  S.add(P.get(1));

  i = 2;
  while (i < n) {
    j = S.size() - 1;
    if (ccw(S.get(j-1), S.get(j), P.get(i)))
      S.add(P.get(i++));
    else S.remove(S.size() - 1);
  }
  return S;
}
```

```
class AngCom implements Comparator<point>{
  public int compare(point a, point b) {
    if (collinear(pivot, a, b))
      return dist(pivot, a) < dist(pivot, b) ? -1 : 1;
    double d1x = a.x - pivot.x, d1y = a.y - pivot.y;
    double d2x = b.x - pivot.x, d2y = b.y - pivot.y;
    return (Math.atan2(d1y,d1x)-Math.atan2(d2y,d2x))<0?-1:1;
  }
}
```