Computational Geometry2

2D Objects: Circles

- **วงกลม Circle** มีจุดศูนย์กลางที่คู่อันดับ (a, b) ใน 2D Euclidean space ด้วยรัศมี **radius** r คือเซตของทุกจุด (x, y) ที่ $(x - a)^2 + (y - b)^2 = r^2$
- ในการตรวจสอบว่าจุดอยู่ภายใน หรือภายนอก หรือบนเส้นรอบวงของวงกลม เราใช้ฟังก์ชันต่อไปนี้ (ทั้งนี้สามารถปรับเป็น floating point ได้)

```
int insideCircle(point_i p, point_i c, int r) { // all integer version
  int dx = p.x - c.x, dy = p.y - c.y;
  int Euc = dx * dx + dy * dy, rSq = r * r; // all integer
  return Euc < rSq ? 0 : Euc == rSq ? 1 : 2; } //inside/border/outside
```

- ค่าคงที่ Pi (π) เป็นอัตราส่วนของเส้นรอบวงกับเส้นผ่านศูนย์กลางของวงกลมนั้นๆ เพื่อป้องกัน error จากการปัด ค่าที่ปลอดภัยสำหรับการแข่งถ้าไม่ได้ระบุค่าคงที่ π มาให้คือ

  pi = acos(-1.0) หรือ pi = 2 * acos(0.0)

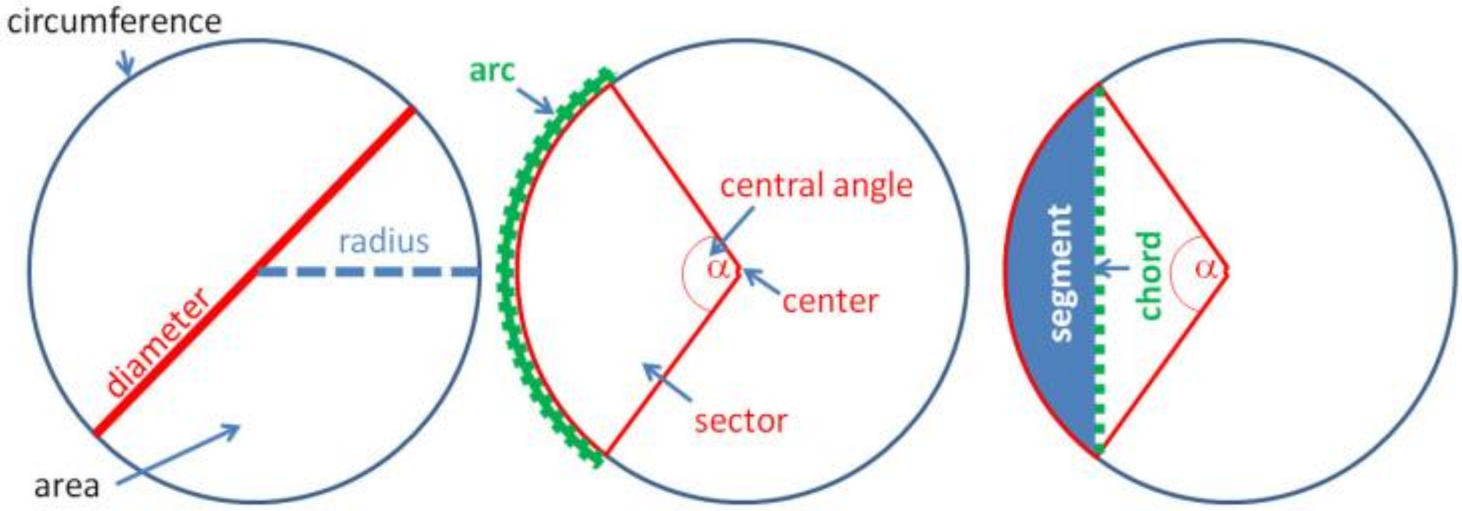
- วงกลมที่มีรัศมี r จะมีเส้นผ่านศูนย์กลาง diameter $d = 2 \times r$ และเส้นรอบวง circumference (or perimeter) $c = 2 \times \pi \times r$

- วงกลมที่มีรัศมี r จะมีพื้นที่ $A = \pi \times r^2$

- **ส่วนโค้งของวงกลม หรือ Arc** นิยามเป็นส่วนที่เชื่อมต่อกันของเส้นรอบวง *c* ของวงกลม เมื่อกำหนดมุมที่จุดศูนย์กลาง central angle α องศา (มุมที่เกิดกับจุดศูนย์กลางวงกลม) เราสามารถคำนวณความยาวของส่วนโค้งวงกลมที่สอดคล้องกันได้จาก

$$arc = \frac{\alpha}{360} * c$$

- **คอร์ด หรือ Chord ของวงกลม** นิยามเป็นส่วนของเส้นตรงที่จุดปลายอยู่บนวงกลม วงกลมรัศมี *r* และมีมุมที่จุดศูนย์กลาง α องศาจะมีคอร์ดที่สอดคล้องกันด้วยความยาว $sqrt(2 \times r^2 \times (1 - cos(\alpha)))$ หรือ $2 \times r \times sin(\alpha/2)$

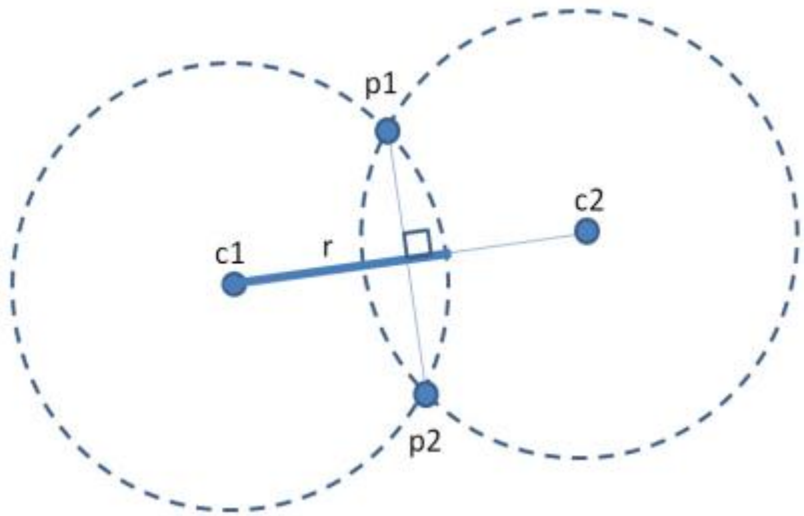


- **เซกเตอร์ หรือ Sector** ของวงกลม นิยามเป็นบริเวณของวงกลมที่ถูกปิดด้วยรัศมี 2 เส้นและส่วนโค้งของวงกลม
- วงกลมที่มีพื้นที่ A จะมีมุมที่จุดศูนย์กลาง α องศา จะมีพื้นที่เซกเตอร์

$$\text{sector area} = \frac{\alpha}{360} * A$$

- **เซกเมนต์ หรือ Segment** ของวงกลม นิยามเป็นบริเวณของวงกลมที่ถูกปิดด้วย chord และ arc
- พื้นที่ของเซกเมนต์หาได้จากการลบพื้นที่ของเซกเตอร์ที่สอดคล้องกับพื้นที่สามเหลี่ยมหน้าจั่วที่แต่ละด้านยาวเป็น: r, r, และความยาว chord
-

- เมื่อกำหนดจุด 2 จุดบนวงกลม (*p*1 และ *p*2) และรัศมี *r* ของวงกลมที่สอดคล้อง เราสามารถหาตำแหน่งของจุดศูนย์กลาง (*c*1 และ *c*2) ของทั้งสองวงกลมได้

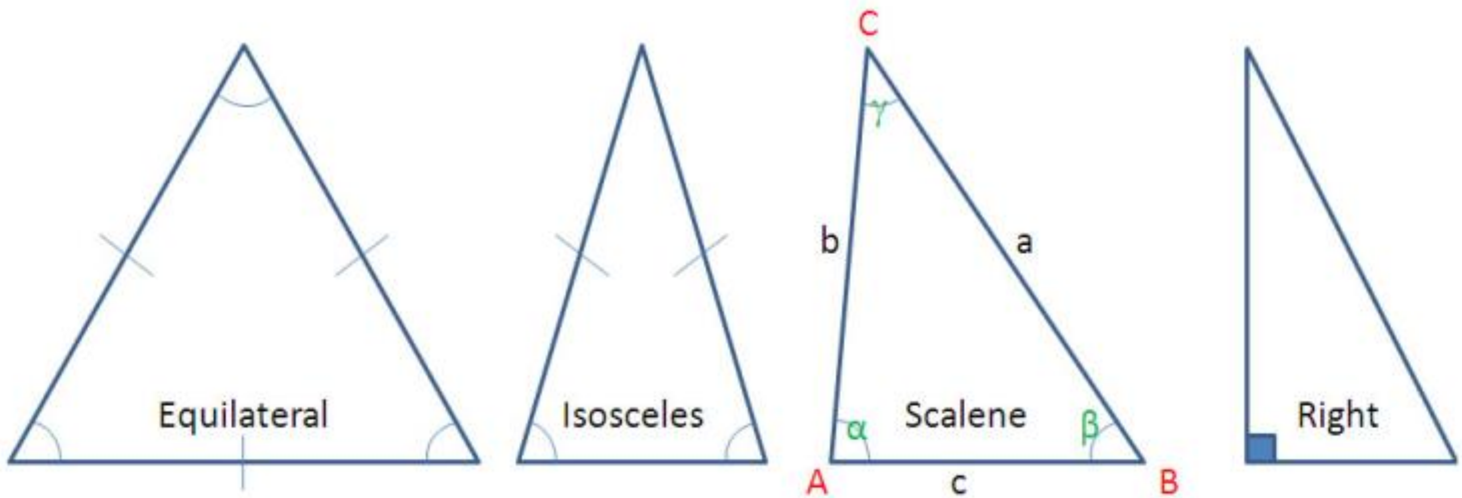


```
bool circle2PtsRad(point p1, point p2, double r, point &c) {
  double d2 = (p1.x - p2.x) * (p1.x - p2.x) +
              (p1.y - p2.y) * (p1.y - p2.y);
  double det = r * r / d2 - 0.25;
  if (det < 0.0) return false;
  double h = sqrt(det);
  c.x = (p1.x + p2.x) * 0.5 + (p1.y - p2.y) * h;
  c.y = (p1.y + p2.y) * 0.5 + (p2.x - p1.x) * h;
  return true; } // to get the other center, reverse p1 and p2
```

https://math.stackexchange.com/questions/1781438/finding-the-center-of-a-circle-given-two-points-and-a-radius-algebraically

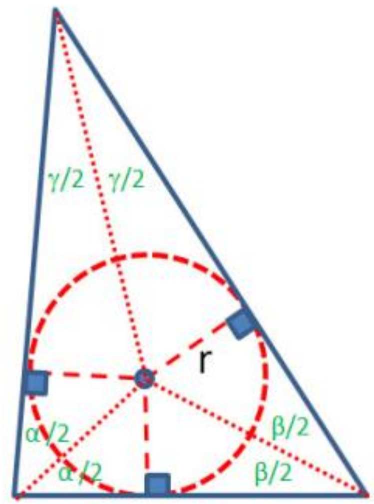
2D Objects: Triangles

- **สามเหลี่ยม Triangle** เป็นรูปหลายเหลี่ยมที่มีจุดมุม 3 จุดและด้าน 3 ด้าน สามเหลี่ยมมีหลายชนิดได้แก่:
  - สามเหลี่ยมด้านเท่า **Equilateral**: ด้านทุกด้านยาวเท่ากันและมีมุมภายใน 60 องศา
  - สามเหลี่ยมหน้าจั่ว **Isosceles**: มีสองด้านที่ยาวเท่ากันและมุมภายในสองมุมเท่ากัน
  - สามเหลี่ยมด้านไม่เท่า **Scalene**: ทุกด้านยาวไม่เท่ากัน
  - สามเหลี่ยมมุมฉาก **Right**: มีมุมภายในมุมหนึ่งเป็นมุม 90 องศา (หรือมุมฉาก **right angle**)



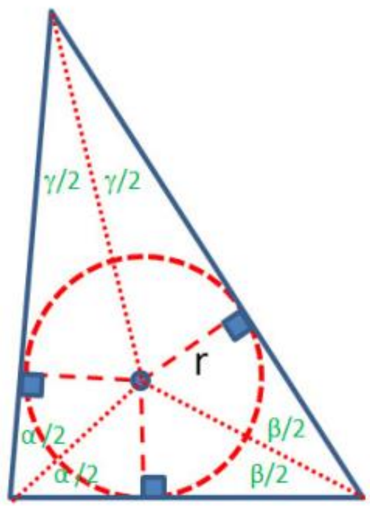
- สามเหลี่ยมที่มีฐานยาว b และสูง h มีพื้นที่ **area** $A = 0.5 \times b \times h$
- สามเหลี่ยมที่มีสามด้าน: a, b, c มีความยาวรอบรูป **perimeter** $p = a + b + c$ และ **semi–perimeter** $s = 0.5 \times p$
- สามเหลี่ยมที่มีสามด้าน: a, b, c และ semi–perimeter s มีพื้นที่ $A = sqrt(s \times (s - a) \times (s - b) \times (s - c))$. สูตรนี้ชื่อว่า **Heron's Formula**.

- สามเหลี่ยมที่มีพื้นที่ A และมี semi-perimeter *s* จะมีวงกลมภายในสามเหลี่ยมที่ใหญ่ที่สุด (**inscribed circle (incircle)**) ที่มีรัศมี $r = A/s$




```
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)); }
```

- จุดศูนย์กลางของ incircle คือจุดบรรจบกันของเส้นแบ่งครึ่งมุมของสามเหลี่ยม

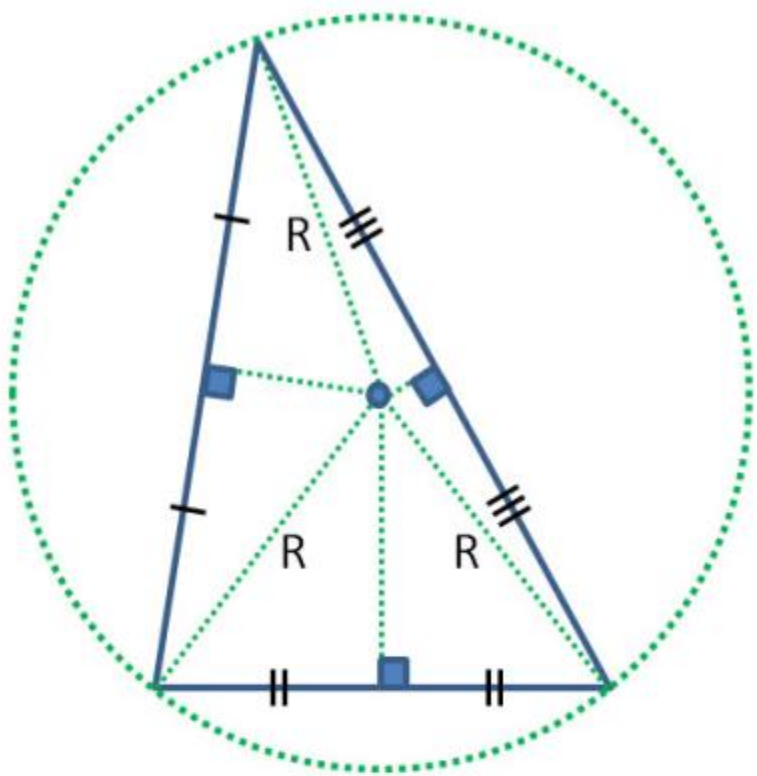


- เราสามารถหาจุดศูนย์กลางได้ ถ้าเรารู้เส้นแบ่งครึ่งมุมสองอัน จากนั้นหาจุดตัด

```
// assumption: the required points/lines functions have been written
// returns 1 if there is an inCircle center, returns 0 otherwise
// if this function returns 1, ctr will be the inCircle center
// and r is the same as rInCircle
int inCircle(point p1, point p2, point p3, point &ctr, double &r) {
  r = rInCircle(p1, p2, p3);
  if (fabs(r) < EPS) return 0; // no inCircle center
  line l1, l2; // compute these two angle bisectors
  double ratio = dist(p1, p2) / dist(p1, p3);
  point 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); // get their intersection point
  return 1; }
```

- สามเหลี่ยมที่มีสามด้าน: *a*, *b*, *c* และมีพื้นที่ *A* จะมีวงกลมที่ล้อมรอบ (**circumscribed circle (circumcircle)**) ที่มีรัศมี

$R = a \times b \times c/(4 \times A)$



```
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)); }
```

- จุดศูนย์กลางของวงกลมที่ล้อมรอบสามเหลี่ยมคือจุดตัดกันระหว่างเส้นแบ่งครึ่งด้านที่ตั้งฉากของสามเหลี่ยม

- ในการตรวจสอบว่าถ้าส่วนของเส้นตรงสามเส้น *a*, *b* และ *c* สามารถทำให้เกิดสามเหลี่ยมได้หรือไม่ เราสามารถตรวจสอบได้ด้วยอสมการ *triangle inequalities*: $(a + b > c)$ && $(a + c > b)$ && $(b + c > a)$

- ถ้าผลที่ได้เป็นเท็จแล้วส่วนของเส้นตรงสามเส้นนั้นไม่สามารถเป็นรูปสามเหลี่ยมได้ ถ้าความยาวของทั้งสามเส้นนำมาเรียงกันโดยให้ *a* เป็นเส้นที่สั้นที่สุดและ *c* เป็นเส้นที่ยาวที่สุดแล้วเราตรวจสอบเพียง $(a + b > c)$

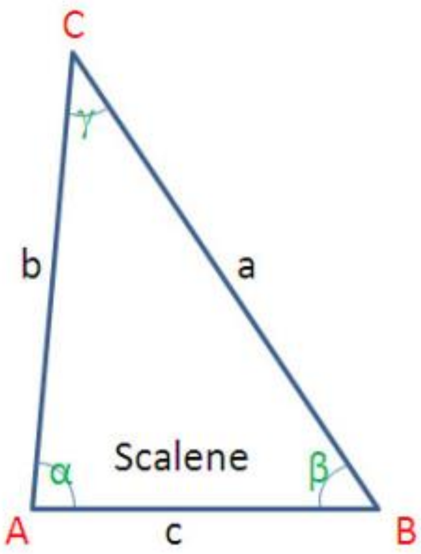
  เมื่อเราศึกษาสามเหลี่ยม ก็ไม่ควรลืมตรีโกณ **Trigonometry**—การศึกษาเกี่ยวกับความสัมพันธ์ระหว่างด้านและมุมระหว่างด้านของสามเหลี่ยม

- ในตรีโกณมิติ กฎของ Cosine (**Law of Cosines** (a.k.a. the **Cosine Formula** or the **Cosine Rule**)) คือบทบัญญัติเกี่ยวกับสามเหลี่ยมทั่วไปที่สัมพันธ์กับความยาวของด้านของมันกับ cosine ของหนึ่งในมุมของมัน
- พิจารณาสามเหลี่ยมด้านไม่เท่า เราจะได้ว่า

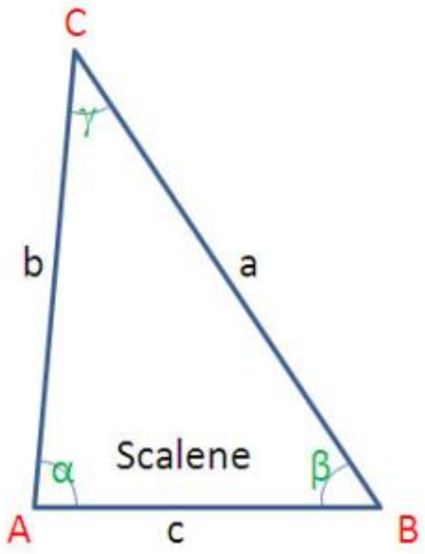
$c^2 = a^2 + b^2 - 2 \times a \times b \times cos(\gamma),$

หรือ $\gamma = acos\left(\frac{a^2+b^2-c^2}{2*a*b}\right).$

สูตรสำหรับมุมอื่นอีกสองมุม α และ β คล้ายกัน



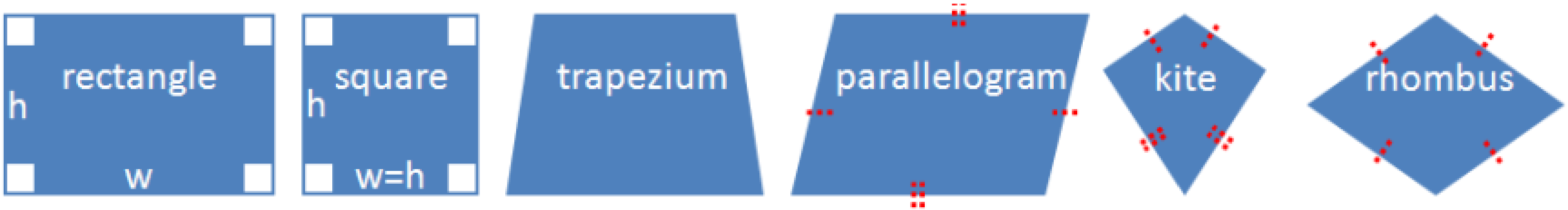
- ในตรีโกณมิติ กฎของ sines (**Law of Sines** (a.k.a. the **Sine Formula** or the **Sine Rule**)) คือสมการที่สัมพันธ์ของความยาวของด้านของสามเหลี่ยมใดๆ ที่มีต่อค่า sine ของมุมของมัน
- ให้ *R* แทนรัศมีของวงกลมล้อมรอบสามเหลี่ยมเราจะได้ว่า
- $\dfrac{a}{\sin(\alpha)} = \dfrac{b}{\sin(\beta)} = \dfrac{c}{\sin(\gamma)} = 2R$



- ทฤษฎีบทปิทากอรัส **Pythagorean Theorem** เป็น special case ของกฎของ Cosines ทฤษฎีนี้ประยุกต์กับมุมฉากเท่านั้น
- ถ้ามุม γ เป็นมุมฉาก (90° or $\pi/2$ radians), แล้ว $cos(\gamma) = 0$, ดังนั้นจากกฎของ Cosines ทำให้ได้: $c^2 = a^2 + b^2$.
- **Pythagorean Triple** คือ triple ของเลขจำนวนเต็มบวกสามจำนวน *a*, *b*, and *c*—โดยปกติเขียน (*a*, *b*, *c*)—ที่ $a^2 + b^2 = c^2$.
- ตัวอย่างที่น่าจะจำกันได้คือ (3, 4, 5).
- ถ้า (*a*, *b*, *c*) เป็น Pythagorean triple, แล้ว (*ka*, *kb*, *kc*) สำหรับเลขจะนวนเต็มบวก k ใดๆ ก็เป็น Pythagorean triple . Pythagorean Triple เป็นการอธิบายความยาวของด้าน 3 ด้านองสามเหลี่ยมมุมฉาก

2D Objects: Quadrilaterals

- **สี่เหลี่ยม Quadrilateral หรือ Quadrangle** เป็นรูปหลายเหลี่ยมมที่มี 4 มุม (มี 4 จุดยอด)



  - **สี่เหลี่ยมมุมฉาก Rectangle** เป็นรูปหลายเหลี่ยมที่มี 4 เส้นเชื่อม 4 จุดยอด และ 4 มุมฉาก สี่เหลี่ยมมุมฉากนั้นมีความกว้าง w และสูง h จะมีพื้นที่ **area** $A = w \times h$ และมีเส้นรอบรูป **perimeter** $p = 2 \times (w + h)$.
  - **สี่เหลี่ยมจัตุรัส Square** เป็น special case ของ สี่เหลี่ยมจัตุรัสที่ $w = h$

- **สี่เหลี่ยมคางหมู Trapezium** เป็นรูปหลายเหลี่ยมที่มี 4 เส้นเชื่อม 4 จุดยอดและมีเส้นเชื่อมคู่หนึ่งขนานกัน ถ้าด้านที่ไม่ขนานกันสองด้านมีความยาวเท่ากัน เราจะเรียกว่า **Isosceles Trapezium** สี่เหลี่ยมคางหมูที่มีด้านที่ขนานกันยาว $w1$ และ $w2$; และมีความสูง h ระหว่างเส้นเชื่อมที่ขนานกันจะมีพื้นที่ $A = 0.5 \times (w1 + w2) \times h$.
- **สี่เหลี่ยมด้านขนาน Parallelogram** เป็นรูปหลายเหลี่ยมที่มี 4 เส้นเชื่อมและมีสี่จุดยอด อีกทั้งด้านตรงข้ามกันจะต้องขนานกัน
- **สี่เหลี่ยมรูปว่าว Kite** เป็นสี่เหลี่ยมที่มีด้านที่ติดยาวเท่ากันสองคู่ พื้นที่ของสี่เหลี่ยมรูปว่าวคือ $diagonal_1 \times diagonal_2/2$.
- **สี่เหลี่ยมขนมเปียกปูน Rhombus** เป็น special case ของ สี่เหลี่ยมด้านขนานที่ทุกด้านยาวเท่ากัน มันยังเป็น special case ของสี่เหลี่ยมรูปว่าวอีกด้วย

Algorithm on Polygon with Libraries

- **Polygon** คือรูปบนระนาบที่ถูกล้อมรอบด้วยเส้นทางปิด (เส้นทางที่เริ่มต้นและจบที่จุดยอดเดียวกัน) ประกอบด้วยลำดับจำกัดของส่วนของเส้นตรง ซึ่งส่วนของเส้นตรงนี้ถูกเรียกว่าขอบหรือด้าน
- จุดที่ด้านสองอันมาบรรจบกันคือจุดยอดของโพลีกอนหรือมุม
- วิธีมาตรฐานในการเก็บ polygon คือการแทนค่าจุดยอดในลำดับวนตามเข็มหรือทวนเข็ม ด้วยจุดยอดแรกเป็นจุดยอดสุดท้าย
- ในที่นี้เราจะใช้ลำดับวนทวนเข็มเป็นหลัก

```
// 6 points, entered in counter clockwise
order, 0-based indexing
vector<point> P;
P.push_back(point(1, 1)); // P0
P.push_back(point(3, 3)); // P1
P.push_back(point(9, 1)); // P2
P.push_back(point(12, 4)); // P3
P.push_back(point(9, 7)); // P4
P.push_back(point(1, 7)); // P5
P.push_back(P[0]); // important: loop back
```

- เส้นรอบรูปของ a polygon ที่มี *n* จุดยอดที่ได้รับมาในบางลำดับ (ว่าจะเป็นตามเข็มหรือทวนเข็ม) สามารถถูกคำนวณได้ตามนี้

```
// returns the perimeter, which is the sum of Euclidian distances
// of consecutive line segments (polygon edges)
double perimeter(const vector<point> &P) {
  double result = 0.0;
  for (int i = 0; i < (int)P.size()-1; i++) // remember that P[0] = P[n-1]
    result += dist(P[i], P[i+1]);
  return result; }
```

Area of a Polygon

- พื้นที่ A ของโพลีกอนของจุดยอด n จุดเมื่อได้รับลำดับบางลำดับมา (ว่าจะเป็นตามเข็มนาฬิกาหรือทวนเข็มนาฬิกา) สามารถถูกคำนวณได้จาก determinant ของ matrix ดังรูป

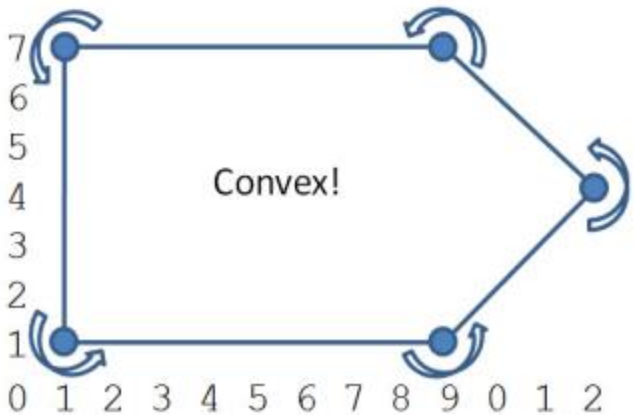
$$A = \frac{1}{2} \times \begin{bmatrix} x_0 & y_0 \\ x_1 & y_1 \\ x_2 & y_2 \\ \dots & \dots \\ x_{n-1} & y_{n-1} \end{bmatrix} = \frac{1}{2} \times (x_0 \times y_1 + x_1 \times y_2 + \ldots + x_{n-1} \times y_0 - x_1 \times y_0 - x_2 \times y_1 - \ldots - x_0 \times y_{n-1})$$

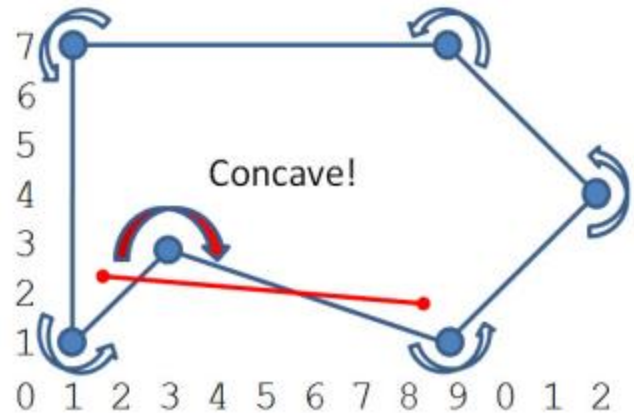
```
// returns the area, which is half the determinant
double area(const vector<point> &P) {
  double result = 0.0, x1, y1, x2, y2;
  for (int i = 0; i < (int)P.size()-1; i++) {
    x1 = P[i].x; x2 = P[i+1].x;
    y1 = P[i].y; y2 = P[i+1].y;
    result += (x1 * y2 - x2 * y1);
  }
  return fabs(result) / 2.0; }
```

- **รูปหลายเหลี่ยมนูน** (convex polygon) คือรูปหลายเหลี่ยมที่มีเนื้อที่ภายในเป็นเซตนูน (convex set) สมบัติต่อไปนี้ของรูปหลายเหลี่ยมเชิงเดียว ซึ่งเทียบเท่าได้กับสมบัติของรูปหลายเหลี่ยมนูน
  - มุมภายในทุกมุมมีขนาดน้อยกว่า 180 องศา
  - ส่วนของเส้นตรงทุกเส้นที่เชื่อมระหว่างจุดยอดสองจุดใดๆ จะวางตัวอยู่ภายในขอบเขตของรูปหลายเหลี่ยม
- รูปที่ไม่ได้เป็นรูปหลายเหลี่ยมนูนจะเรียกว่าเป็น **รูปหลายเหลี่ยมเว้า** (concave polygon)

Checking if a polygon is convex

- รูปหลายเหลี่ยมจะถูกบอกว่าเป็น **Convex** ถ้าส่วนของเส้นตรงใดๆ ที่วาดภายในรูปหลายเหลี่ยมไม่ตัดกับขอบใดๆ ของรูปหลายเหลี่ยม ถ้าไม่เช่นนั้นรูปหลายเหลี่ยมจะถูกเรียกว่า **Concave.**





- อย่างไรก็ตามในการทดสอบว่า polygon เป็น convex มีวิธีการคำนวณแบบง่ายกว่า “การลองตรวจสอบทุกส่วนของเส้นตรงว่าถ้าทุกส่วนของเส้นตรงวาดแล้วอยู่ภายใน polygon”

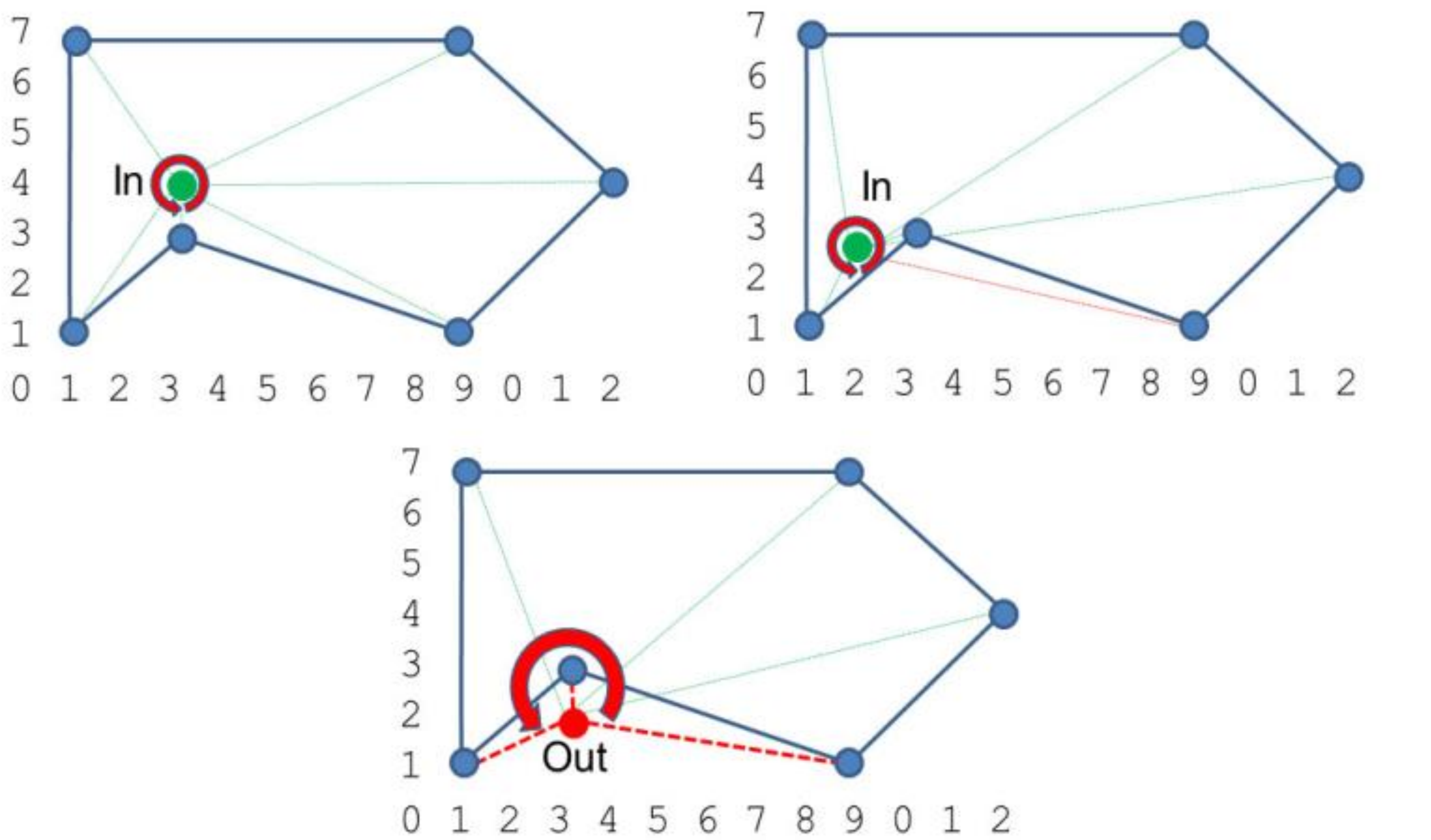
- เราเพียงแค่ตรวจสอบว่าทุกจุดยอดสามอันที่ติดกันของนั้นหมุนไปทางเดียวกัน (หมุนไปทางซ้าย/ทวนเข็ม ถ้าโหนดถูกเรียงในลำดับทวนเข็ม หรือหมุนไปทางขวาในอีกกรณีหนึ่ง

- ถ้าเราพบว่ามีอย่างน้อยสามจุดที่เป็นเท็จแล้ว polygon นั้นเป็น concave

```cpp
bool isConvex(const vector<point> &P) { // returns true if all three
  int sz = (int)P.size(); // consecutive vertices of P form the same turns
  if (sz <= 3) return false; // a point/sz=2 or a line/sz=3 is not convex
  bool isLeft = ccw(P[0], P[1], P[2]); // remember one result
  for (int i = 1; i < sz-1; i++) // then compare with the others
    if (ccw(P[i], P[i+1], P[(i+2) == sz ? 1 : i+2]) != isLeft)
    return false; // different sign -> this polygon is concave
  return true; }
```

Checking if a Point is Inside a Polygon

- การทดสอบอีกอย่างที่ถูกทำกับ polygon P คือการตรวจสอบว่าจุด pt อยู่ภายในหรือภายนอก polygon P.
- ฟังก์ชันต่อไปนี้ ใช้ 'winding number algorithm' มันทำงานโดยการคำนวณผลรวมของมุมระหว่างจุด 3 จุด: $\{P[i], pt, P[i+1]\}$ เมื่อ $(P[i]–P[i+1])$ คือด้านที่ติดกันของ polygon P, ต้องระวังหากหมุนซ้ายจะเพิ่มมุม หากหมุนขวาจะลดมุม
- ถ้าผลรวมได้ 2π (360 degrees), แล้ว pt อยู่ภายใน polygon P

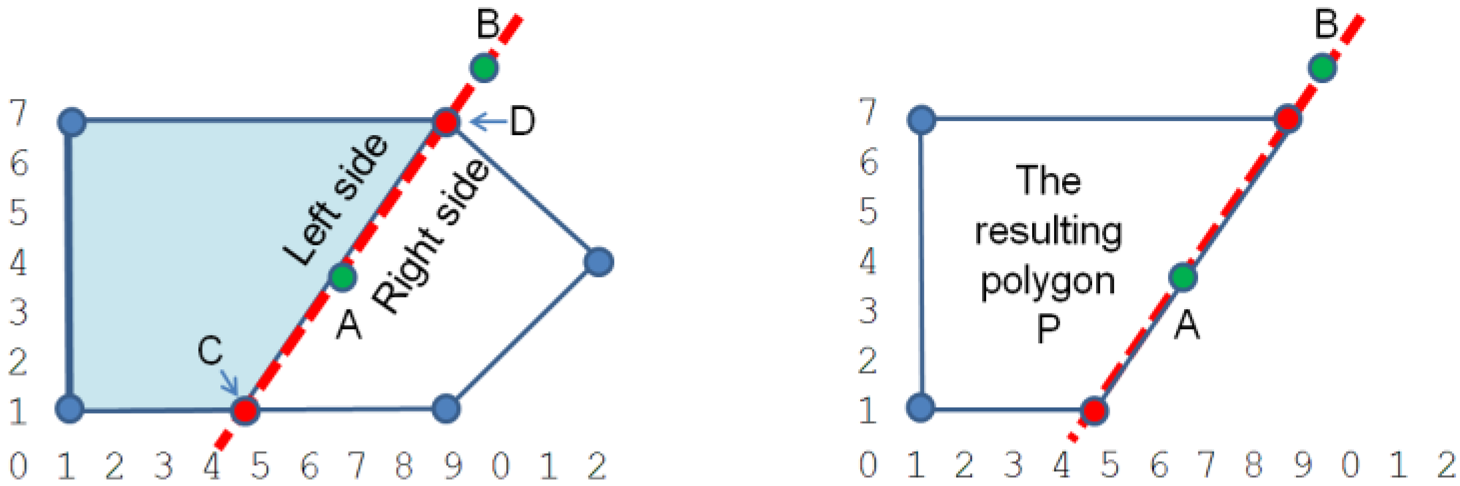


In
In
Out

```
// returns true if point p is in either convex/concave polygon P
bool inPolygon(point pt, const vector<point> &P) {
  if ((int)P.size() == 0) return false;
  double sum = 0;//assume the first vertex is equal to the last vertex
  for (int i = 0; i < (int)P.size()-1; i++) {
    if (ccw(pt, P[i], P[i+1]))
      sum += angle(P[i], pt, P[i+1]); // left turn/ccw
    else sum -= angle(P[i], pt, P[i+1]); } // right turn/cw
  return fabs(fabs(sum) - 2*PI) < EPS; }
```

Cutting Polygon with a Straight Line

- สิ่งที่น่าสนใจอีกอย่างที่เราทำกับ *convex* polygon คือการตัดมันเป็นสอง convex sub–polygons ด้วยเส้นตรงที่นิยามด้วยจุดสองจุด a และ b.
- แนวคิดเบื้องต้นของ cutPolygon routine คือการวนรอบทีละจุดยอดของ original polygon Q ทีละจุด
- ถ้าเส้น ab และจุดยอด v ของ polygon ให้เกิดหมุนซ้าย (นั่นหมายความว่า v อยู่ทางซ้ายของเส้น ab), เราจะเก็บ v ไว้ใน new polygon P.
- เมื่อเราหาด้านของ polygon edge ที่ตัดกับเส้น ab, เราจะใช้จุดตัดนั้นเป็นส่วนหนึ่งของ new polygon P จากนั้นเราจะข้ามจุดยอดของ Q ที่อยู่ทางด้านขวาของเส้น ab. จนกระทั่งเรากลับมายังอีกด้านของ polygon ที่ตัดกับเส้น ab อีกครั้ง

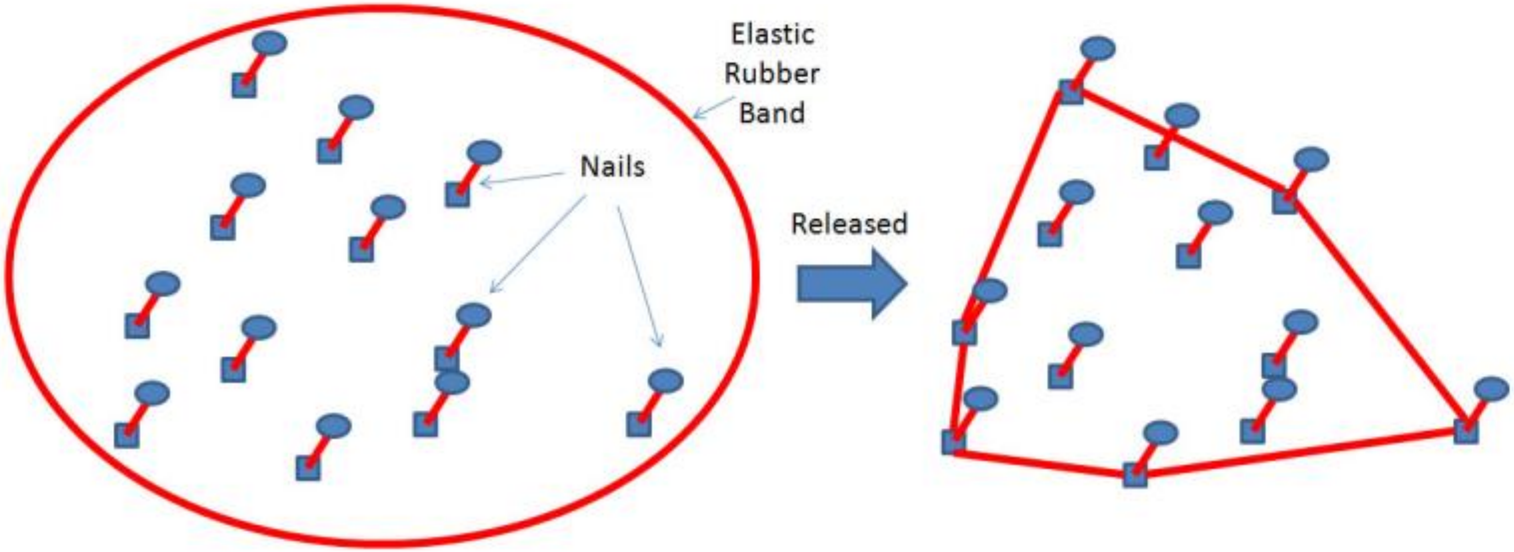

B
D
Left side
Right side
A
C
7
6
5
4
3
2
1
0 1 2 3 4 5 6 7 8 9 0 1 2
B
The resulting polygon P
A
7
6
5
4
3
2
1
0 1 2 3 4 5 6 7 8 9 0 1 2

```
// line segment p-q intersect with line A-B.
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 = fabs(a * p.x + b * p.y + c);
  double v = fabs(a * q.x + b * q.y + c);
  return point((p.x * v + q.x * u) / (u+v), (p.y * v + q.y * u) / (u+v)); }
```

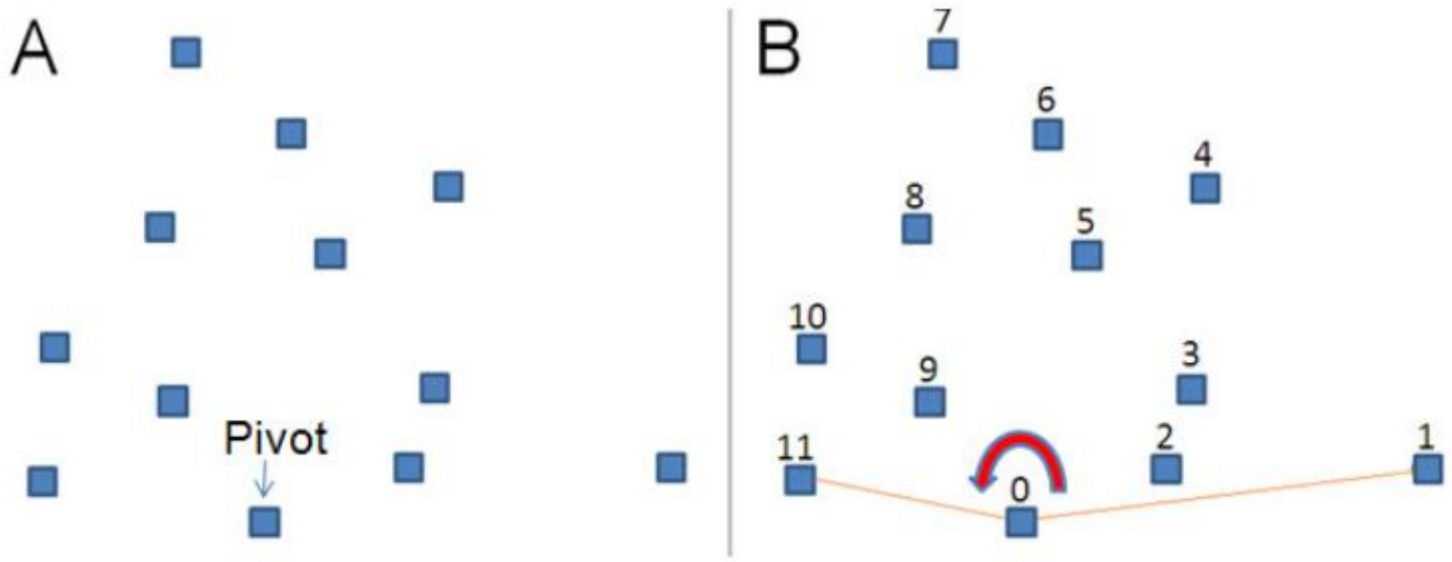
```
// cuts polygon Q along the line formed by point a -> point b
// (note: the last point must be the same as the first point)
  vector<point> cutPolygon(point a, point b, const vector<point> &Q) {
  vector<point> P;
  for (int i = 0; i < (int)Q.size(); i++) {
    double left1 = cross(toVec(a, b), toVec(a, Q[i])), left2 = 0;
    if (i != (int)Q.size()-1) left2 = cross(toVec(a, b), toVec(a, Q[i+1]));
    if (left1 > -EPS) P.push_back(Q[i]); // Q[i] is on the left of ab
    if (left1 * left2 < -EPS) // edge (Q[i], Q[i+1]) crosses line ab
    P.push_back(lineIntersectSeg(Q[i], Q[i+1], a, b));
  }
  if (!P.empty() && !(P.back() == P.front()))
    P.push_back(P.front()); // make P's first point = P's last point
  return P; }
```

Finding the Convex Hull of a Set of Points

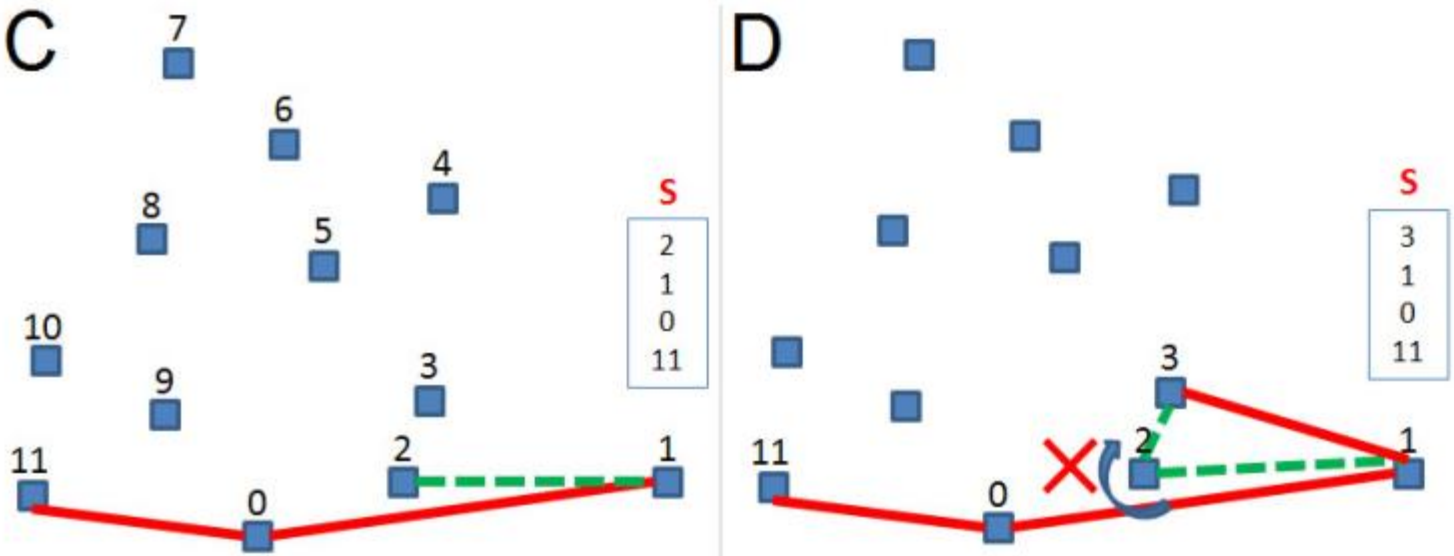
- **Convex Hull** ของเซตของจุด *P* คือ convex polygon ที่มีขนาดเล็กที่สุด *CH*(*P*) สำหรับแต่ละจุดใน *P* นั้นเป็นขอบของ *CH*(*P*) หรือไม่ก็อยู่ภายใน
- จินตนาการว่ามีจุดอยู่บนกระดาน แล้วเราตอกตะปูบนจุด จากนั้นนำเอาหนังยางมาครอบจุด
- ถ้าปล่อยหนังยาง หนังยางมันจะหุ้มเป็นบริเวณขนาดเล็กที่สุดเท่าที่จะทำได้ บริเวณที่หุ้มได้นั่นคือ convex hull ของเซตของจุด
- ในการหา convex hull ของเซตของจุดนั้นมี application ในชีวิตจริงคือ ปัญหา *packing* problems.


Elastic
Rubber
Band
Nails
Released

- เนื่องจากทุกจุดยอดใน $CH(P)$ คือจุดยอดในเซต P, algorithm ในการหา convex hull คืออัลกอริทึมที่ตัดสินใจว่าจุดใดใน P ที่ควรเลือกเป็นส่วนหนึ่งของ convex hull.
- มีอัลกอริทึมในการหา convex hull อยู่หลายอัน ในเรื่องสุดท้ายนี้เราเลือก$O(n \log n)$ Ronald *Graham's Scan* algorithm
- Graham's scan algorithm เริ่มต้นจะเรียงทั้ง n จุดของ P เมื่อจุดแรกไม่ต้องทำซ้ำกับจุดสุดท้าย โดยเรียงตามมุมเมื่อเทียบกับจุด pivot.
- ในตัวอย่างต่อไป เราจะหยิบจุดล่างขวาสุดเป็น pivot
- หลังจากเรียงตามมุมแล้ว, เราจะเห็นเส้น 0–1, 0–2, 0–3, ..., 0–10, และ 0–11 ในลำดับทวนเข็ม

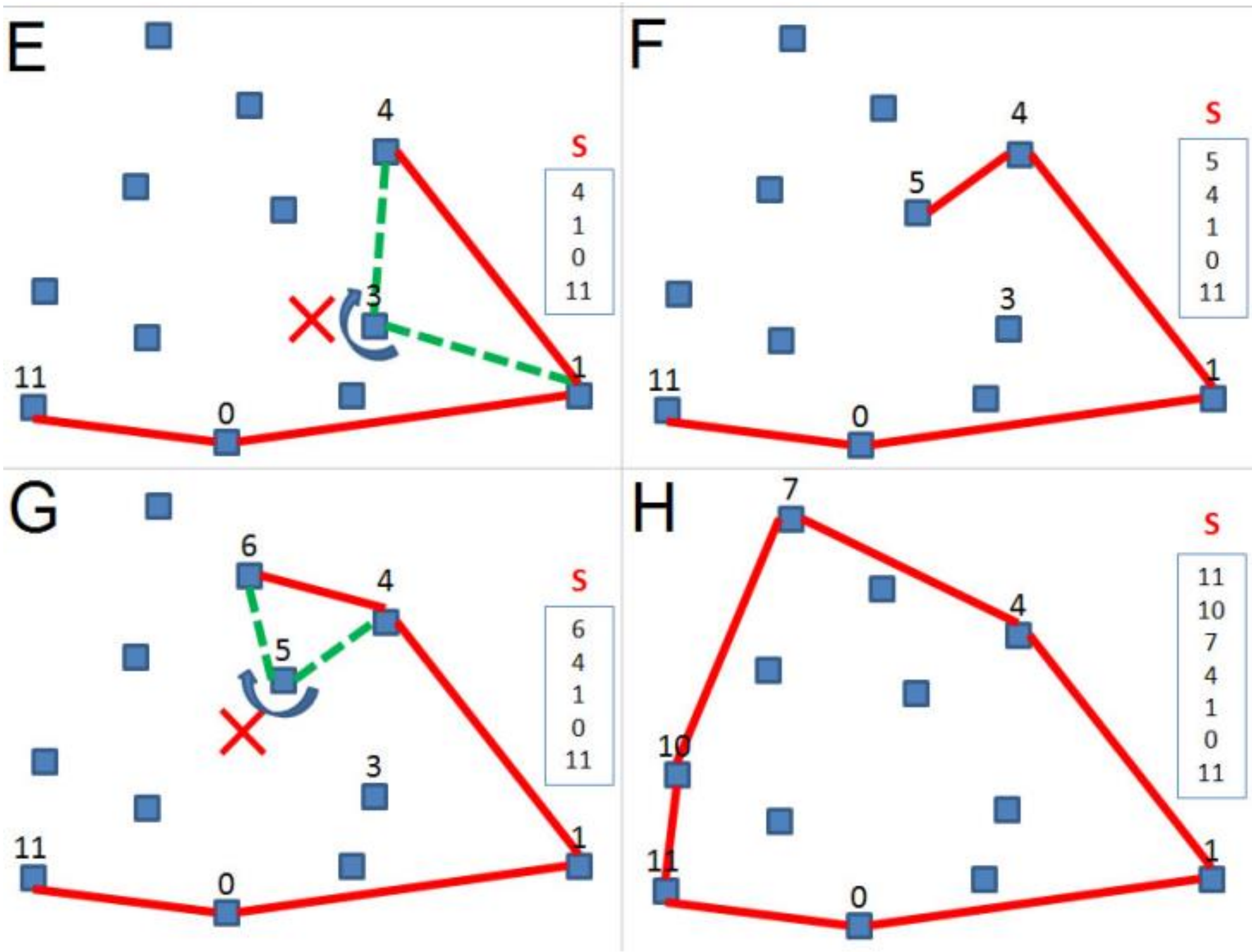

A
Pivot
B
7
6
4
8
5
10
3
9
11
0
2
1

- จากนั้น algorithm จะเก็บ stack *S* ของจุด แต่ละจุดของ *P* จะถูก push เพียงครั้งเดียวเข้าไปใน *S* และจุดที่เป็นส่วนหนึ่งใน *CH*(*P*) will be จะถูก pop ออกจาก *S*.
- Graham's Scan จะรักษาคุณสมบัติต่อไปนี้: สมาชิก 3 ตัวบนของ stack *S* จะต้องหมุนซ้ายตลอด (ซึ่งเป็นคุณสมบัติพื้นฐานของ convex polygon)
- เริ่มต้นเราจะใส่จุด 3 จุดได้แก่ จุด *N*–1, 0, และ 1.
- ในตัวอย่าง stack เริ่มต้นจะเก็บจุด 11–0–1 ซึ่งจะเป็นหมุนซ้ายเสมอ
- ต่อไปก็จะลองใส่จุด 2 และ 0–1–2 เป็นหมุนซ้ายดังนั้นเราจะเก็บจุด 2 เข้าไปใน Stack ดังนั้น *S* ขณะนี้จะเก็บ (bottom) 11–0–1–2 (top)



- ต่อไปรูปทางขวาเราจะลองใส่จุด 3 และ 1–2–3 เป็นการหมุนขวา ถ้าเราเก็บจุดที่มาก่อนก่อนจุด 3(จุด 2) จะทำให้ไม่ได้ convex polygon ดังนั้นเราจะ pop จุด 2 ออกจาก Stack ทำให้ *S* ปัจจุบันเป็น (bottom) 11–0–1 (top) อีกครั้ง จากนั้นเราลองใส่จุด 3 อีกครั้งครั้งนี้ 0–1–3 ดังนั้นเราจะเก็บจุด 3 ทำให้ Stack *S* ปัจจุบันเป็น (bottom) 11–0–1–3 (top)

- เราจะทำเช่นนี้ไปเรื่อยๆ



- เมื่อ Graham's Scan หยุด อะไรก็ตามที่เหลืออยู่ใน S จะคือจุดใน $CH(P)$ ในตัวอย่างคือ (bottom) 11–0–1–4–7–10–11 (top))
- Graham Scan's จะกำจัดทุกจุดที่หมุนขวา เมื่อจุดยอดสามจุดที่ติดกันใน S จะหมุนซ้ายเสมอ ทำให้สุดท้ายเราได้ convex polygon
-

```
point pivot(0, 0);
bool angleCmp(point a, point b) { // angle-sorting function
  if (collinear(pivot, a, b)) // special case
    return dist(pivot, a) < dist(pivot, b); // check which one is
closer
  double d1x = a.x - pivot.x, d1y = a.y - pivot.y;
  double d2x = b.x - pivot.x, d2y = b.y - pivot.y;
  return (atan2(d1y, d1x) - atan2(d2y, d2x)) < 0; } // compare two
angles
```

```
vector<point> CH(vector<point> P) { // the content of P may be reshuffled
  int i, j, n = (int)P.size();
  if (n <= 3) {
    if (!(P[0] == P[n-1])) P.push_back(P[0]); // safeguard from corner case
    return P; } // special case, the CH is P itself
// first, find P0 = point with lowest Y and if tie: rightmost X
  int P0 = 0;
  for (i = 1; i < n; i++)
    if (P[i].y < P[P0].y || (P[i].y == P[P0].y && P[i].x > P[P0].x))
      P0 = i;
  point temp = P[0]; P[0] = P[P0]; P[P0] = temp; // swap P[P0] with P[0]
// second, sort points by angle w.r.t. pivot P0
  pivot = P[0]; // use this global variable as reference
  sort(++P.begin(), P.end(), angleCmp); // we do not sort P[0]
// third, the ccw tests
  vector<point> S;
  S.push_back(P[n-1]); S.push_back(P[0]); S.push_back(P[1]); // initial S
  i = 2; // then, we check the rest
  while (i < n) { // note: N must be >= 3 for this method to work
    j = (int)S.size()-1;
    if (ccw(S[j-1], S[j], P[i])) S.push_back(P[i++]); // left turn, accept
    else S.pop_back(); } // or pop the top of S until we have a left turn
  return S; } // return the result
```

- ใน Main

P = CH(P);