計算幾何入門

~多角形~

アルゴリズム研究室 D1

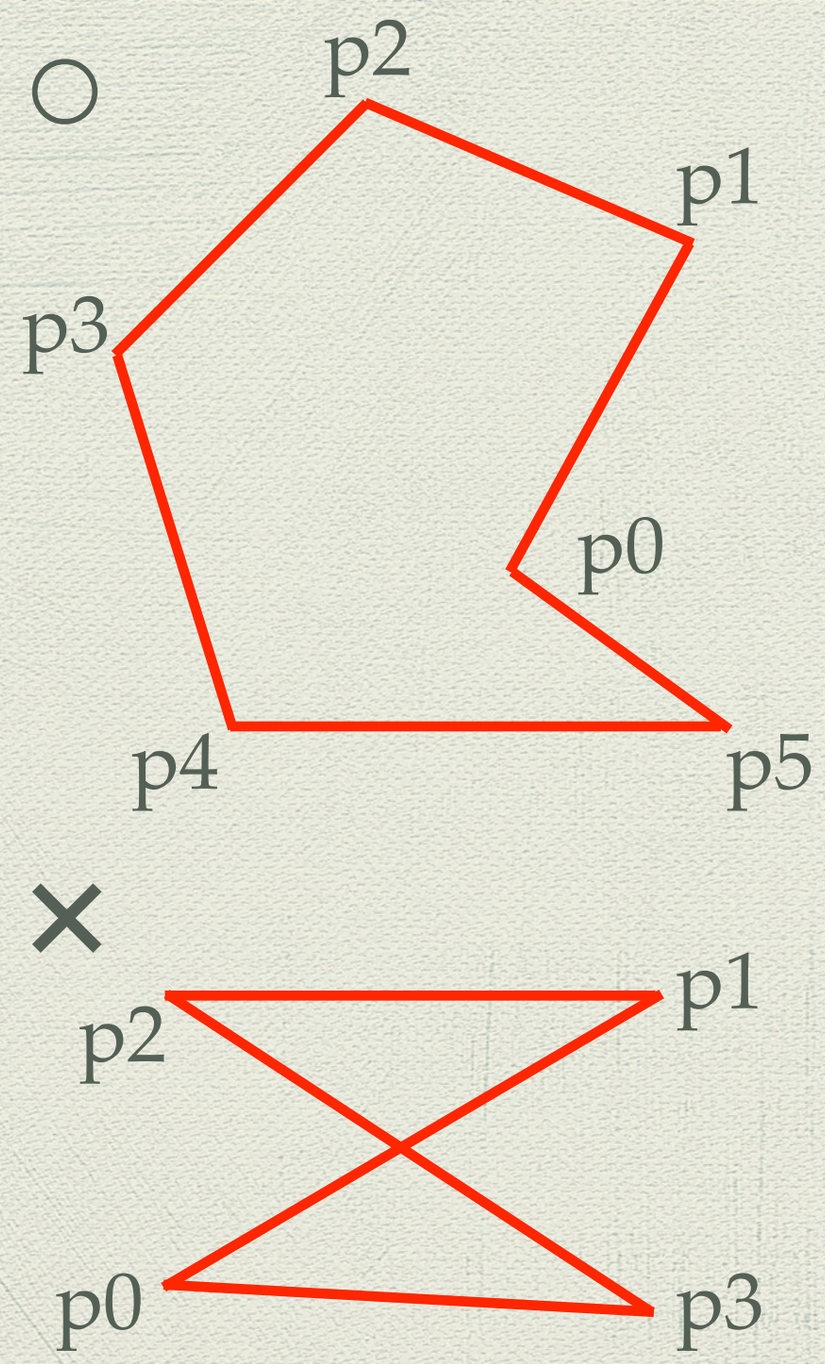
井上 祐馬

内容

- 多角形の表現
- 多角形の面積
- 多角形に対する点の内外判定
- 点集合の凸包
- 最遠点対問題

多角形の表現

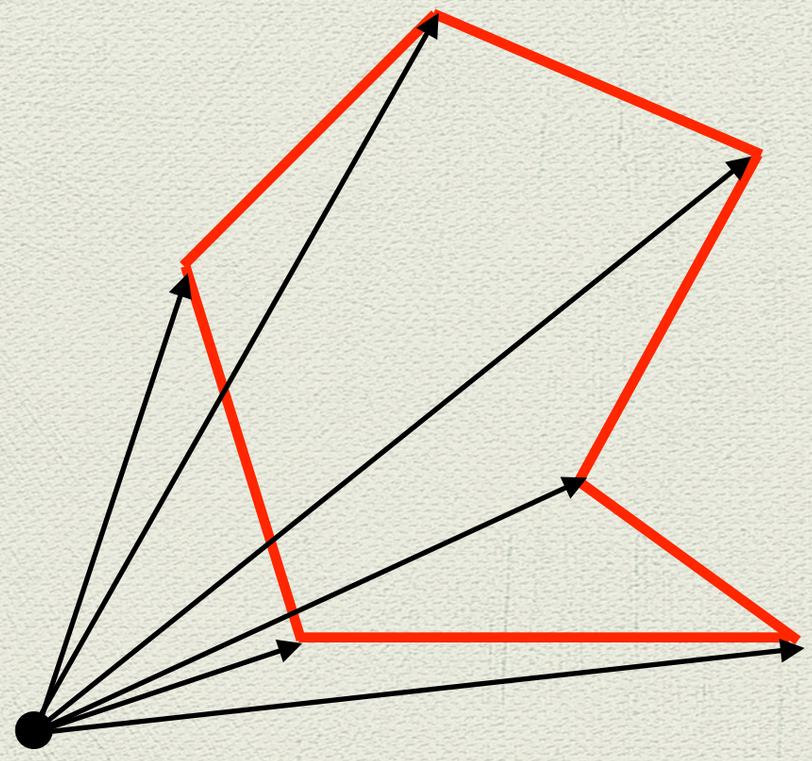
- 頂点を反時計回りに辿った列で表す
  - typedef vector<P> Poly;
- 頂点を順番に結んだ線分集合が辺になる
- 通常の問題では自己交差はない



多角形の面積

- 多角形の面積 = 任意の点pから隣接する2点へのベクトルの外積の総和 / 2
- 計算量： O(n)

```
double poly_area(Poly p){
  if(p.size()<3)return 0;
  double res = cross(p[p.size()-1], p[0]);
  for(int i=0;i<(int)p.size()-1;i++){
      res += cross(p[i], p[i+1]);
  }
  return res/2;
}
```



多角形に対する点の内外判定

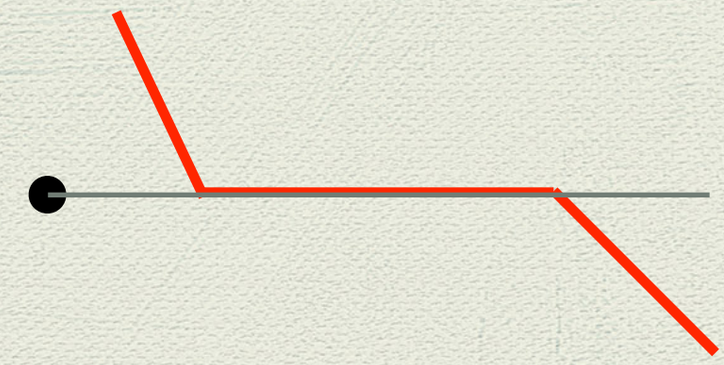
- 点が多角形の内部にある ⇔ 点から伸ばした半直線と多角形の交差回数が奇数回
- 計算量： O(n)

```
bool in_poly(Poly p, P x){
 int n = p.size();
 double xMax = x.real();
 for(int i=0;i<n;i++){
  if(xMax < p[i].real())xMax = p[i].real();
  if(EQ(x,p[i]))return false;
 }
 L h = L( x,P(xMax + 1.0, x.imag()) );
 //pを通る半直線を、x軸に平行で多角形の最大x座標
    より大きい座標までの線分とする

.....
```

多角形に対する点の内外判定

- 点が多角形の内部にある ⇔ 点から伸ばした半直線と多角形の交差回数が奇数回
- 計算量： O(n)

```
bool in_poly(Poly p, P x){
  ......

  int c = 0;
  for(int i=0;i<n;i++){
    L l = L(p[i],p[(i+1)%s]);
    if(!para(h, l) && is_cp(h, l)){
      P cp = line_cp(h, l);
      if(cp.real() < x.real() + EPS)continue;
      if(EQ(cp, (l.fs.imag() < l.sc.imag())?l.sc:l.fs))c++;
    }
  }
  return (c&1)?true:false;
}
```



やばいケース

・平行に交差する

→ カウントしない

・線分上に点pが乗っている

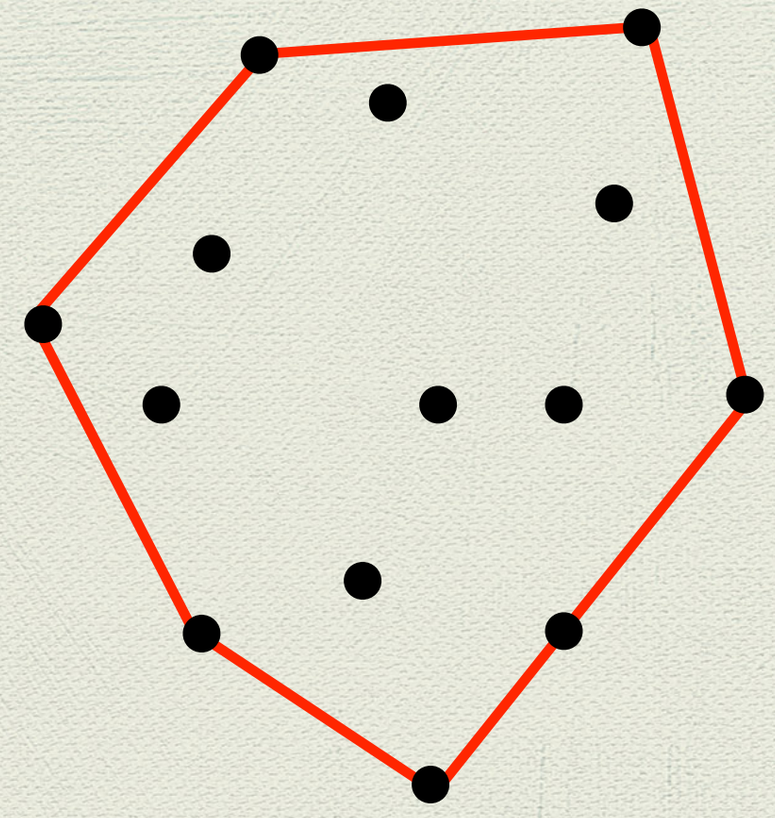
→ 問題の扱いに依る

・ちょうど頂点で交わる

→ 上の端点と交差している ときだけカウントする

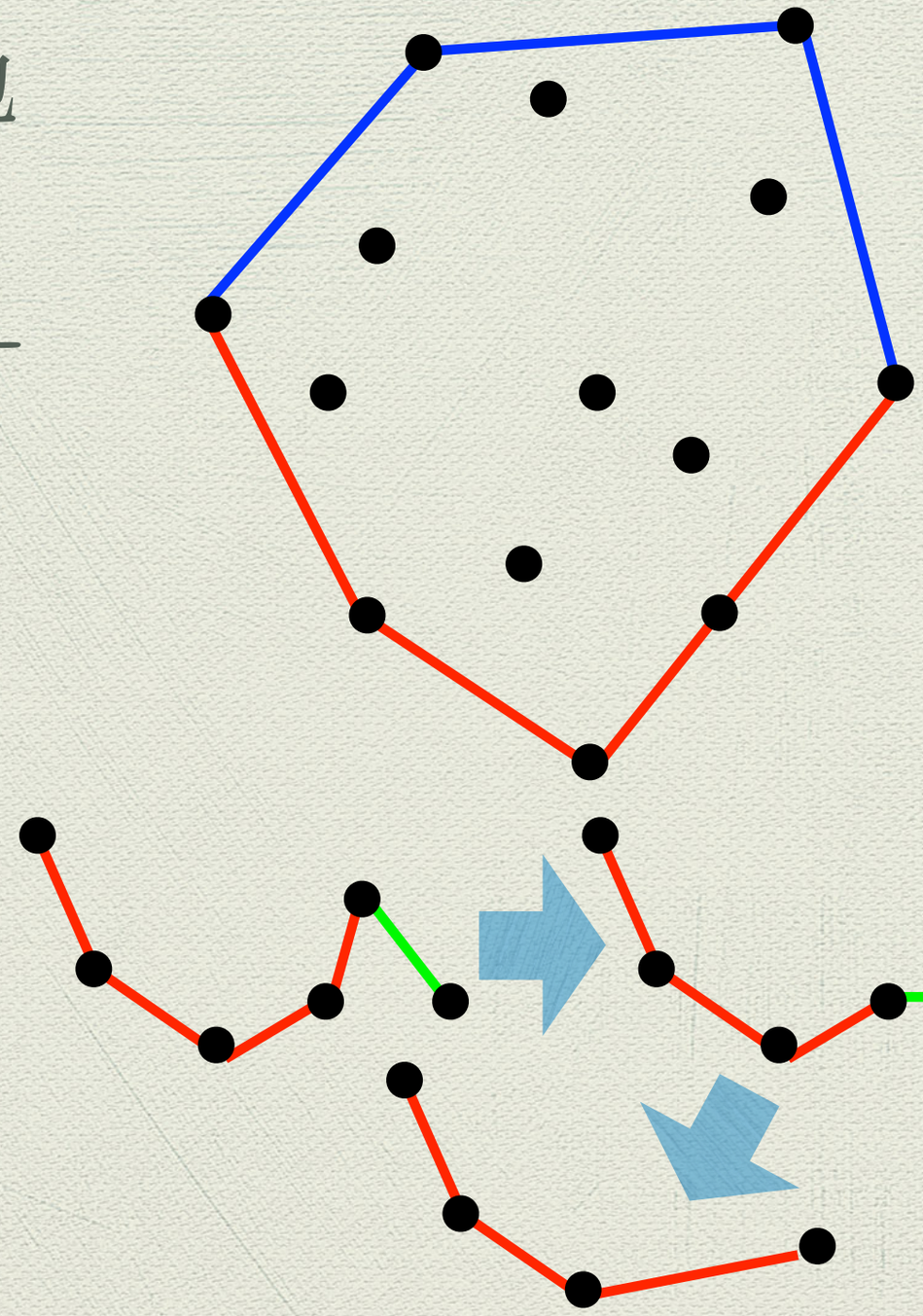
点集合の凸包

- 凸多角形：すべての内角が180°(=π)以下の多角形
  - よい性質がたくさんある
- 凸包：与えられた点集合Sの部分集合Pを頂点とする凸多角形で、Sに含まれるすべての点を内部、もしくは境界上に持つ凸多角形



凸包を求めるアルゴリズム

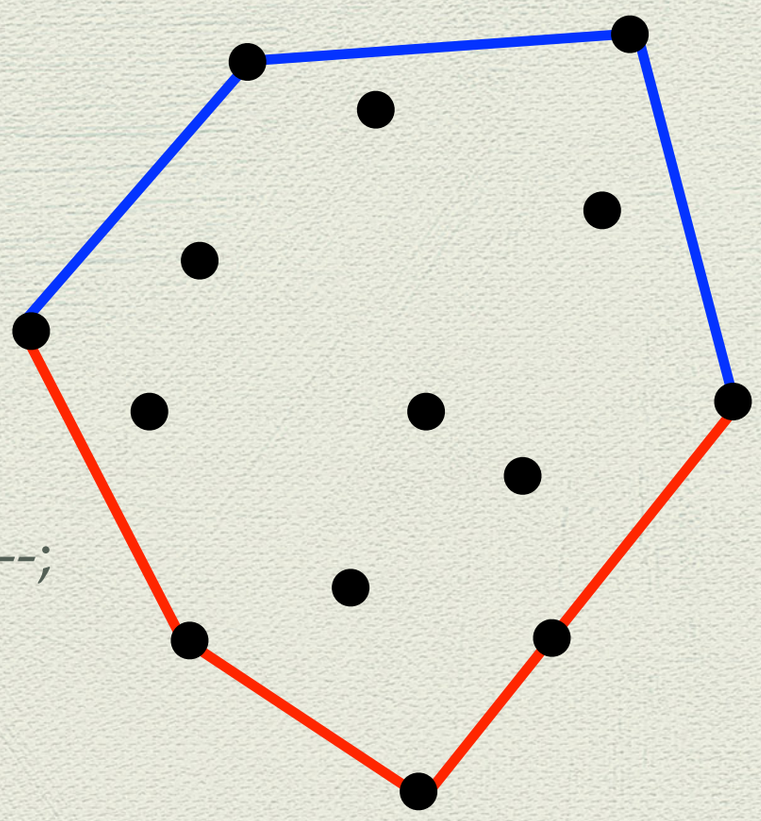
- 最左の点と最右の点は必ず凸包に含まれる
- 下凸と上凸をそれぞれ求めてマージすればよい
- 下凸：点をx座標ソートし、左から順に暫定凸包に加えて行く
- 前の2点とのなす角がπを超えたら、1つ前の頂点を暫定凸包から削除

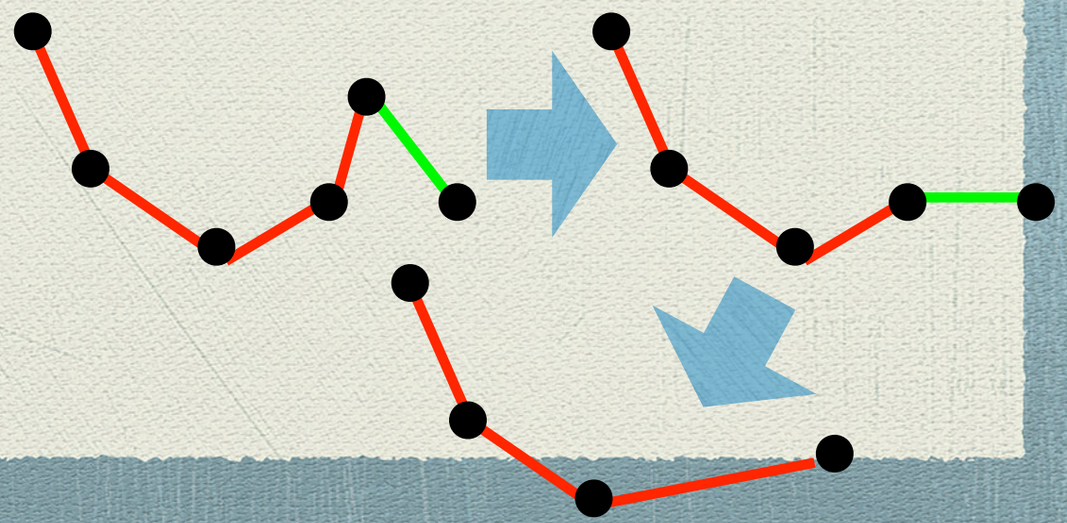


凸包の実装

計算量：ソート O(n logn) + 暫定凸包の更新 O(n)

```
Poly convex_hull(vector<P> v){
  int n = v.size(), k = 0;
  sort(v.begin(), v.end());
  Poly r(2*n);
  for(int i=0;i<n;i++){
    while(k>1 && cross(r[k-1]-r[k-2],v[i]-r[k-1]) <= EPS)k--;
    r[k++] = v[i];
  }
  for(int i=n-2,t=k;i>=0;i--){
    while(k>t && cross(r[k-1]-r[k-2],v[i]-r[k-1]) <= EPS)k--;
    r[k++] = v[i];
  }
  r.resize(k-1);
  return r;
}
```



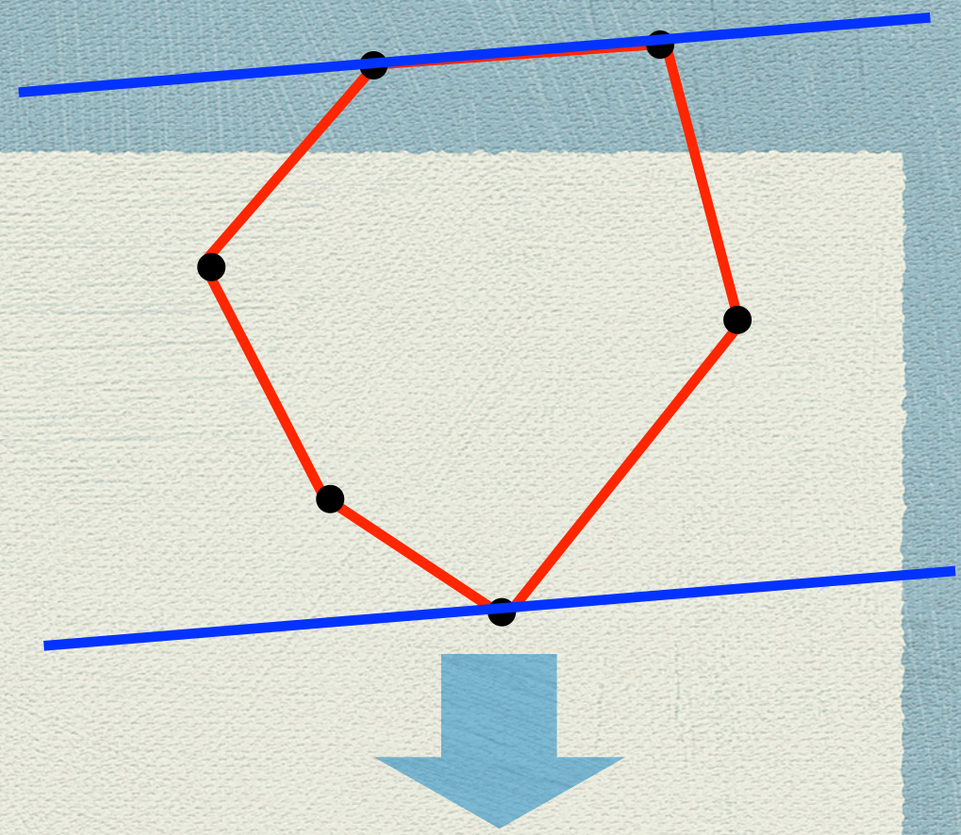


最遠点対問題

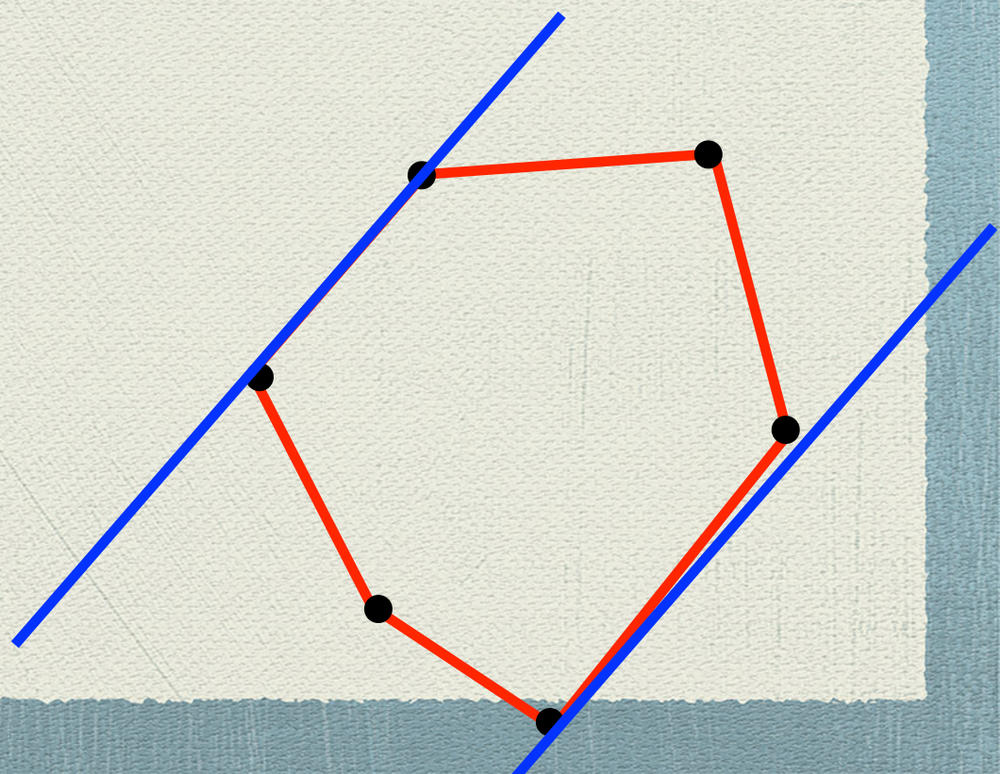
- 与えられた点集合Sのうち、最も離れた2点間の距離を求める問題
- 普通にやるとO(n^2)かかる
- O(n logn)で計算可能
  → 凸包 + キャリパー法

キャリパー法

- 凸包の最遠点対を求めるアルゴリズム
  - 点集合の最遠点対＝凸包の最遠点対



- 2つの平行な直線をノギスのように回転させる
- 2直線間の距離が最も離れたところ＝最遠点対間の距離



キャリパー法の実装

- 計算量： O(n)

```
double caliper(Poly p){
  int n = p.sz;
  if(n<=1)return 0;
  if(n==2)return abs(p[0]-p[1]);

  int i = 0, j = 0;
  for(int k=0;k<n;k++){
    if(!(p[i]<p[k]))i = k;  //左下をiにする
    if(p[j]<p[k])j = k;     //右上をjにする
  }

  double res = 0;
  int si = i, sj = j;
  while(i != sj || j != si){
    res = max(res,abs(p[i]-p[j]));
    if( cross(p[(i+1)%n] - p[i], p[(j+1)%n] - p[j]) < 0)i = (i+1)%n;
    else j = (j+1)%n;
  }
  return res;
}
```