Program Promenade
連載

大小の贈り物

田中 哲朗（東京大学情報基盤センター）
ktanaka@tanaka.ecc.u-tokyo.ac.jp

■問題の定義

今回取り上げる問題は2000年決勝大会のProblem D「Gifts Large and Small」である．

包装専門会社が多角形の贈り物を長方形の包装材で包むサービスを行っている．なるべく狭い（面積の小さい）長方形で包みたい客と，贈り物が長方形の各辺に触れるという条件で，なるべく広い（面積の大きい）長方形で包みたい客の両方の要望に答えるプログラムを作成するというのが題名の由来になっている．

図-1の三角形では，図の左側の向きに置いて包装すると面積が最小になり，図の右側の向きに置いて包装すると面積が最大になる．平行移動しても面積は変わらないので，回転の角度だけによって面積が決まる．

入力は多角形の定義の繰り返しで与えられる．多角形の定義の最初の1行は，頂点の数n($3 \le n \le 100$)となっている．その後にn行の頂点の定義が続く．各頂点は整数のペアで表される．最後に頂点数0の多角形が現れると，入力は終了となる．以下に問題文中のサンプル入力を引用する．

```
3
-3  5
7   9
17  5
4
10 10
10 20
20 20
20 10
0
```

出力としては問題番号の後に最小面積と最大面積を小数点以下3桁までを表示することが求められる．さきほどのサンプル入力に対する出力は以下のようになる．

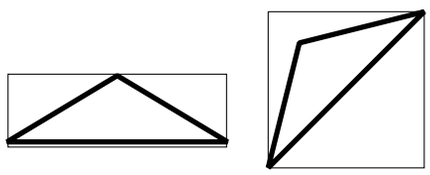

図-1　問題例

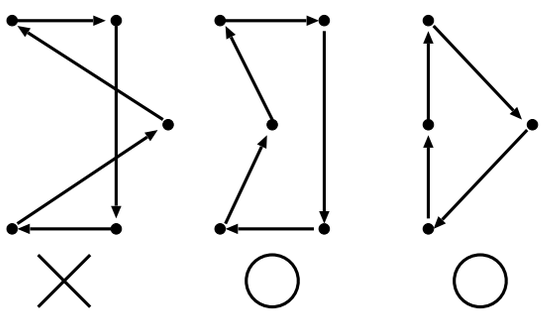

図-2　制約を満たす多角形

```
Gift 1
Minimum area = 80.000
Maximum area = 200.000

Gift 2
Minimum area = 100.000
Maximum area = 200.000
```

多角形は自己交叉（図-2の左のようなもの）がないこと，面積が0でないこと，時計回り順に並んでいることが保証されている．凸多角形とは断っていないので，図-2の中央のように凹んだ部分があっても構わない．また，図-2の右のように，尖っていない点が頂点として与えられることもあり得る．

■素朴な解法

今回は解法を示すプログラム言語としてはC++を使う．ただし，標準的なヘッダファイル等の記述は省くことにする．

まずは，準備のためにいくつかのクラスと関数を定義しておく．まず，点を表す構造体を定義する．頂点の座標は整数の対で与えられるが，回転すると常に整数になるとは限らないので，doubleの対で表す．

```
struct Point{
  double x,y;
  Point(double xx,double yy) :x(xx),y(yy){}
};
```

原点を中心にθ回転させる回転行列R_θは

$$R_\theta = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}$$

となるので，これを適用する関数rotateも定義する．

```
Point rotate(Point const& p,
             double th){
  return Point(p.x*cos(th)-p.y*sin(th),
               p.x*sin(th)+p.y*cos(th));
}
```

多角形はvector<Point>で表す．多角形を角度θ回転させたときの包装紙の面積は以下の関数で求めることができる．

```
double calcArea(vector<Point> const& ps,
                double th){
  // コンテストのときは，バグをなるべく早く発見
  // するために，assert は使った方がよい
  assert(ps.size()>0);
  // 最初の点
  Point p=ps[0].rotate(th);
  // 左右端のx座標
  double minx= p.x, maxx=p.x;
  // 上下端のy座標
  double miny= p.y, maxy=p.y;
  // 残りの点について更新
  for(size_t i=1;i<ps.size();i++){
    Point p=ps[i].rotate(th);
    minx=min(minx,p.x);
    maxx=max(maxx,p.x);
    miny=min(miny,p.y);
    maxy=min(maxy,p.y);
  }
  return (maxx-minx)*(maxy-miny);
}
```

ほかに，面積の最大値と最小値を保持する構造体MinMaxも準備しておく．

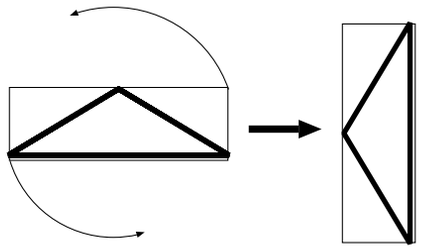

図-3　π/2回転させても面積は同じ

```
// 最大値と最小値を保持
struct MinMax{
  double minVal,maxVal;
  MinMax(const double mn,const double mx)
   :minVal(mn),maxVal(mx){}
  void update(MinMax const& mm){
    minVal=min(minVal,mm.minVal);
    maxVal=max(maxVal,mm.maxVal);
  }
};
// 2つの値から作る
MinMax minMax2(double v0, double v1){
  return MinMax(min(v0,v1),max(v0,v1));
}
```

さて，これで準備は整った．まず，誰でも考えつくと思われる以下の素朴な解法を試してみる．

- 多角形を角度θ回転させたときの包装紙の面積は容易に求まる．
- θを0からπ/2までの範囲で角度δずつ増やしていき，面積の最小値と最大値を更新していく．

図-3のように，π/2回転させても面積は同じなのでθは0からπ/2までの範囲で動かせばよいことに注意されたい．

多角形を表すvector<Point>型のpointsを引数にとって，最大最小面積をMinMax型で返す関数calcMinMaxは以下のように書ける．

```
MinMax calcMinMax(
        vector<Point> const& ps,
        double delta){
  double val=calcArea(ps,0.0);
  MinMax ret(val,val);
  for(double th=delta;th<= M_PI_2 ;
      th+=delta){
    double area=calcArea(ps,th);
    ret.update(MinMax(area,area));
  }
  return ret;
}
```

deltaとして大きな値を与えると，プログラムは早

く終了するが，最大値，最小値を与える θ とのずれが大きくなるため，誤差も大きくなる．逆に小さな値を与えると，誤差は小さくなるが，プログラムの実行時間は増える．

サンプル入力に対しては，10^{-3} 程度の値に設定すると，小数点以下 3 桁まで正しい値が返っているが，どんな問題でも正しい解答を得るためには，delta としてはどのような値を与えればよいだろうか．座標値の範囲として 0 ～ 99 と指定されているならば，10^{-7} 程度を与えれば，有効桁数は 7 桁程度となり，小数点以下 3 桁程度まで正しく求まる．

しかし，問題を読み直してみると，座標値は整数とだけ定義され，範囲が指定されていないことに気づく．プログラミング言語の常識に基づいて，32 ビット符号付き整数の範囲で表現できる整数であれば，何が来ても文句は言えない．

よく考えると，32 ビット符号付き整数の何が来ても正しい結果を返すプログラムを作るのは容易ではないことが分かる．たとえば，

```
4
0 0
0 1000000001
1000000001 1000000001
1000000001 0
```

という正方形を与えてみる．最小面積は何も回転しないときで，面積は 1000000002000000001.000 となるはずだが，この数を正確に表現するには通常の 64 ビット浮動小数点数の有効桁数では不足している．

long double を使うと SPARC 等では 112 ビット，x86 では 64 ビットの仮数部があるので，正しい結果になるが，long double の精度に関しては規格では保証されていない．かといって，より精度を確保できる，多倍長整数同士の割算，あるいは連分数表現で解答することを求めているとも考えにくい．

そこで，ここでは最小値，最大値を与える θ に十分近い double の値で計算したときに，double の範囲内で真の値に十分近い値を計算することを目標にすることにする．これで，本当に出題者側の求めるレベルに達しているかは確認できないが，おそらく問題ないだろう．

この基準で，さきほどのプログラムを見直してみると，最小値，最大値を与える θ に十分近い double を求めるためには，delta として，10^{-16} 程度を与えなければいけないことが分かる．これではコンテストの審判用マシンはおろか，世界最速のスーパーコンピュータを持ってきても，制限時間内に終わらないことは明らかなので，ほかの解法を考える必要がある．

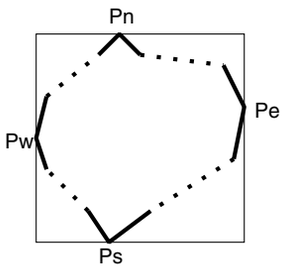

図 -4　p_e, p_n, p_w, p_s の定義

■高校の数学

面積が最大値，最小値となるのはどのような角度の場合だろうか．まずは，上下左右端となる頂点が入れ替わる角度が候補となり得る．

一方，区間 (θ_0, θ_1) で上下左右端となる点が変わらないとしたとき，面積はどう変化するのだろうか．これは，高校レベルの問題なので簡単に解くことができる．**図 -4** のように，この区間で右端となる点の（$\theta=0$ のときの）座標を $p_e=(x_e, y_e)$, とし，同様に上端，左端，下端を p_n, p_w, p_s とする[☆1]．

$(\theta_0 \le \theta \le \theta_1)$ 回転させたときの長方形の面積は，$a=x_e-x_w$, $b=y_e-y_w$, $c=x_n-x_s$, $d=y_n-y_s$ とすると，

$$S(\theta)=(a\cos\theta-b\sin\theta)(c\sin\theta+d\cos\theta)$$

となる．

$t=\tan\theta$ とする受験数学のテクニックを使うと，$\cos\theta=1/\sqrt{1+t^2}$, $\sin\theta=t/\sqrt{1+t^2}$ なので，

$$S(t)=\frac{\{-bct^2+(ac-bd)t+ad\}}{t^2+1}$$

という簡単な式になる．t で微分して整理すると，

$$\frac{\mathbf{d}S}{\mathbf{d}t}=\frac{(bd-ac)t^2-2(bc+ad)t-(bd-ac)}{(t^2+1)^2}$$

となる．

分母は常に正で，分子は高々二次なので，簡単に解ける．$A=bd-ac$, $B=bc+ad$ とすると，$0 \le t$ の範囲で $\mathbf{d}S/\mathbf{d}t=0$ となるのは，以下のようになる．

$$\begin{cases} 0 & if\ A=0 \\ (B+\sqrt{B^2+A^2})/A & if\ A>0 \\ (B-\sqrt{B^2+A^2})/A & if\ A<0 \end{cases}$$

この t を θ に戻して，(θ_0, θ_1) の区間内にあれば，極値であることが分かる．

それではプログラムを書いてみよう．まずは，上下左右端がどの点に対応するかを保持するクラスを定義

☆1　East, North, West, South の頭文字をとっている．

する．

```
enum Dir {
  East=0, North=1, West=2, South=3
};
// 多角形の東西南北の端が多角形の
// 何番目の点に対応するかを保持
struct ENWSIndex{
  int indices[4];
  ENWSIndex(int e,int n,int w,int s){
    indices[East]=e; indices[North]=n;
    indices[West]=w; indices[South]=s;
  }
};
```

さきほどは面積を計算するだけだった，calcArea を面積と ENWSIndex の対を返すように変更する．

```
// calcArea の返り値
// 面積と ENWSIndex の対
struct Ans {
  double area;
  ENWSIndex enwsIndex;
  Ans(double a,const ENWSIndex& enws)
    :area(a),enwsIndex(enws){}
};

// points を theta 傾けたときの Ans
Ans calcArea(vector<Point> const& ps,
             double theta){
  assert(ps.size()>0);
  Point p=rotate(ps[0],theta);
  double minx= p.x, maxx=p.x;
  double miny= p.y, maxy=p.y;
  int e=0, n=0, w=0, s=0;
  for(size_t i=1;i<ps.size();i++){
    Point p=rotate(ps[i],theta);
    if(p.x<minx){ w=i; minx=p.x; }
    else if(p.x>maxx){ e=i; maxx=p.x; };
    if(p.y<miny){ s=i; miny=p.y; }
    else if(p.y>maxy){ n=i; maxy=p.y; };
  }
  return Ans((maxx-minx)*(maxy-miny),
             ENWSIndex(e,n,w,s));
}
```

$\theta_0 \le \theta \le \theta_1$ で，左右上下端が変わらないとしたときに，この区間の最大最小値を求める関数 calcSameEdges を定義する．これは，先ほどの極値を求める式をほとんど書き下しただけである．

```
MinMax calcSameEdges(
         vector<Point> const& ps,
         double th0,double th1){
  Ans ans0=calcArea(ps,(th0+th1)/2.0);
  Point pe=ps[ans0.enwsIndex.indices[East]];
  Point pn=ps[ans0.enwsIndex.indices[North]];
  Point pw=ps[ans0.enwsIndex.indices[West]];
  Point ps=ps[ans0.enwsIndex.indices[South]];
  double a=pe.x-pw.x, b=pe.y-pw.y;
  double c=pn.x-ps.x, d=pn.y-ps.y;
  double t0=tan(th0);
  double t1=tan(th1);
  double area0=(a-b*t0)*(c*t0+d)/(1+t0*t0);
  double area1=(a-b*t1)*(c*t1+d)/(1+t1*t1);
  double A= (b*d-a*c), B2=(a*d+b*c);
  if(A==0){ // th=0 で極値を取る
    return minMax2(area0,area1);
  }
  double sqrtD=sqrt(B2*B2+A*A);
  double t;
  if(A<0) t= (B2-sqrtD)/A;
  else t=(B2+sqrtD)/A;
  // 範囲外の場合
  if(t<tan(th0) || tan(th1)<t)
    return minMax2(area0,area1);
  double area=(a-b*t)*(c*t+d)/(1+t*t);
  return MinMax(min(area,min(area0,area1)),
                max(area,max(area0,area1)));
}
```

後は，左右上限端が入れ替わる角度を求めればよい．p_0 と p_1 が入れ替わる角度は，以下のようにして求まる．

```
double theta90(Point const& p0,
               Point const& p1){
  double dx=p1.x-p0.x;
  double dy=p1.y-p0.y;
  return fmod(M_PI-atan2(dy,dx),M_PI_2);
}
```

頂点の数は高々 100 個なので，頂点の対すべてについて，$0 \le \theta \le \pi/2$ の範囲で，x 軸あるいは y 軸に平行になる角度を求めればよい．大部分の角度では左右上限端は入れ替わらないが，すべての頂点の対の組合せは高々 4950 通りなので，大したことはない．プログラムは以下のようになる．

```
MinMax calcAll(vector<Point> const& ps){
    vector<double> ths;
    ths.push_back(0.0);
    ths.push_back(M_PI_2);
    for(size_t i=0;i<ps.size();i++)
      for(size_t j=i+1;j<ps.size();j++){
        ths.push_back(theta90(ps[j],
                              ps[i]));
      }
    sort(ths.begin(),ths.end());
    MinMax ret=calcSameEdges(ps,ths[0],
                             ths[1]);
    for(size_t i=1;i<ths.size()-1;i++){
```

```
      ret.update(calcSameEdges(ps,ths[i],
                               ths[i+1]));
    }
    return ret;
}
```

現在の計算機を用いれば，頂点数 100 個の問題であっても一瞬で求まる．

■凸閉包の利用

前章のプログラムでも，プログラムコンテストの条件をクリアするという面では十分である．しかし，効率という面では不要な角度についても計算してしまうため，問題が多い．頂点数を 10000 まで増やしてみると，途端に止まらなくなってしまう．しかし，ちょっとした改良で，効率が大きく改善する．

まず，問題の前処理として，**図 -5** のように，多角形の凸閉包 (convex hull) を取る．凸閉包を取っても，包装紙の面積は変わりないことが容易に分かる．一方，凸多角形であれば，上下左右端が次に変化する角度の決定も容易になる．

2 次元平面上の点の集合に対して，凸閉包を求めるアルゴリズムとしては，Graham[1] によって効率の良いアルゴリズムが考案されている．Graham のアルゴリズムは，前半でソートを必要とするので，計算量は要素数を N として，$O(N \log N)$ となるが，今回の問題では，元の点を順に辿っていくと時計回り順に並んだ自己交叉のない多角形になっているという条件があるので，それを略すことができ，$O(N)$ で実現できる．

まずは，**図 -6** のように，p_0 から p_1 に向かうベクトルに対して，p_2 が右側（時計回り）にあるときに 1，左側（時計の反対回り）にあるときは -1 となる関数 ccw を定義する．これは，幾何アルゴリズムの常として，丸暗記しておいてもよいし，忘れてしまったときは，回転行列と内積からその場で求めても大したことはない．

```
int ccw(Point const& p0, Point const& p1,
        Point const& p2){
  double dx1=p1.x-p0.x, dy1=p1.y-p0.y;
  double dx2=p2.x-p0.x, dy2=p2.y-p0.y;
  double r90_dx2= -dy2, r90_dy2=dx2;
  double product=dx1*r90_dx2+dy1*r90_dy2;
  if(product<0.0) return -1;
  else if(product==0.0) return 0;
  else return 1;
}
```

まずは，必ず凸となる頂点を探す．ここでは，x が

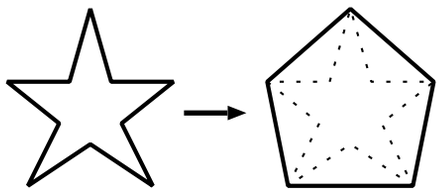

図 -5　凸閉包（**convex hull**）

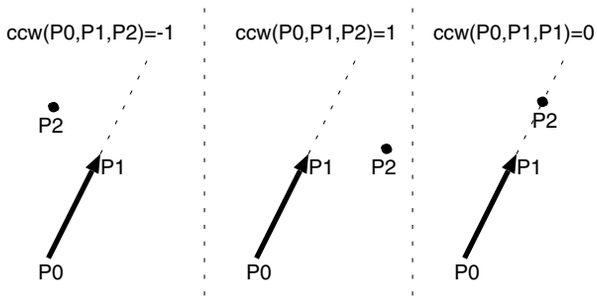

図 6　関数 **ccw** の定義

最大となる頂点 (複数有り得る) の中で y が最大のものを選んでいる．そこから，順に頂点を辿っていき，ccw が負になるところが見つかったら，中央の頂点を削除する．

```
void make_convex_hull(
       vector<Point>& points){
  assert(points.size()>2);
  // 必ず凸な点を探す
  double maxx=points[0].x;
  double maxx_y=points[0].y;
  int maxxIndex=0;
  int srcSize=points.size();
  for(int i=1;i<srcSize;i++){
    if(points[i].x>maxx ||
      (points[i].x==maxx &&
       points[i].y>maxx_y)){
      maxxIndex=i;
      maxx=points[i].x;
      maxx_y=points[i].y;
    }
  }
  vector<Point> newPoints;
  int srcIndex=maxxIndex;
  newPoints.push_back(points[srcIndex]);
  srcIndex=(srcIndex+1)%srcSize;
  newPoints.push_back(points[srcIndex]);
  srcIndex=(srcIndex+1)%srcSize;
  int dstIndex=1;
  for(;;){
    Point p=points[srcIndex];
    while(ccw(newPoints[dstIndex-1],
              newPoints[dstIndex],p)<=0){
      newPoints.pop_back();
```

```
      dstIndex--;
      if(dstIndex==0)break;
    }
    if(srcIndex==maxxIndex)break;
    newPoints.push_back(p);
    dstIndex++;
    srcIndex=(srcIndex+1)%srcSize;
  }
  points.swap(newPoints);
}
```

この前処理が済めば，後は先ほどのプログラムで $n(n-1)$ 通り調べた角度を，隣接する頂点に対してのみ行えばよい．以下のようになる．

```
MinMax calcAll(vector<Point> const& ps){
    vector<double> ths;
    ths.push_back(0.0);
    ths.push_back(M_PI_2);
    for(size_t i=0;i<ps.size();i++){
      ths.push_back(
        theta90(ps[(i+1)%ps.size()],
                ps[i]));
    }
    sort(ths.begin(),ths.end());
    MinMax ret=calcSameEdges(ps,ths[0],
                             ths[1]);
    for(size_t i=1;i<ths.size()-1;i++){
      ret.update(calcSameEdges(ps,ths[i],
                               ths[i+1]));
    }
    return ret;
}
```

このプログラムなら，頂点数が 10000 でも問題なく求めることができる．

■数値計算に基づく解法

前章のプログラムで一応の完成品といえるが，自分が解答者だとしたら，極値を求めるために，微分して式を簡単にするあたりで，「自分はプログラミングコンテストの問題に取り組んでいるのか，受験数学の問題を解いているのか」と考え，憂鬱になってくるだろうし，途中の式の変形のあたりで，ミスをして結局は間違ったプログラムを書いてしまう可能性が高い．

さきほどの calcSameEdges を微分した結果を使わずに二分法で極値を求めるようにしてみる．区間 (θ_0, θ_1) 内で極値が高々 1 個だと仮定すると，区間内に極値が含まれる場合は，区間の端での面積の微係数の符号が異なることが分かる．微係数は単純に $\theta_0+\delta$, $\theta_1-\delta$ での面積を元に求めることができるが，δ が小さすぎると精度の関係で差が出なかったり，係数の符号が数値誤差のために逆になってしまう可能性があるので，面積の差が小数点以下 4 桁程度になるまで大きくしていくことにする．

```
const double delta=1e-15;
MinMax calcSameEdges(
         vector<Point> const& ps,
         double th0,double th1){
  Ans ans0=calcArea(ps,th0);
  Ans ans1=calcArea(ps,th1);
  if(th1-th0<1e-14)
    return minMax2(ans0.area,ans1.area);
  Ans ans0_d=ans0,ans1_d=ans1;
  for(double d=delta;d<th2-th0;d*=2){
    ans0_d=calcArea(ps,th0+d);
    if(abs(ans0.area-ans0_d.area)>1e-4)
      break;
    }
  }
  for(double d=delta;d<th1-th2;d*=2){
    ans1_d=calcArea(ps,th1-d);
    if(abs(ans1.area-ans1_d.area)>1e-4){
      break;
    }
  }
  if((ans0_d.area-ans0.area) *
     (ans1.area-ans1_d.area) >0)
    return minMax2(ans0.area,ans1.area);
  MinMax ret=
    calcSameEdges(ps,th0,th2);
  ret.update(
        calcSameEdges(ps,th2,th1));
  return ret;
}
```

手計算でミスをする可能性が高いが，プログラムを作る際にはミスが少ない人の場合は，手計算の部分の負担をなるべく減らして，上のようなプログラムを作るのもよいだろう．

参考文献

1）Graham, R. L.: An Efficient Algorithms for Determining the Convex Hull of a Finite Planar Set, Information Processing Letters, 1(1972).

（平成 15 年 2 月 10 日受付）