Programar es crear

Álvaro Martínez Echevarría

<ame@acm.org>

«Regalos pequeños y grandes»: solución

El enunciado de este problema apareció en el número 150 de Novática (marzo-abril 2001, p. 73). Es el programa D de los planteados en el 24º Concurso Internacional de Programación de la ACM (2000)

```
/ *
* La solución al problema «Regalos pequeños y grandes» es
* relativamente larga y compleja, aunque la idea es simple: ir girando
* cada polígono hasta que alguno de sus lados sea vertical u horizontal.
* Cuando eso ocurre, la expresión de la función que describe el área
* de la caja se modifica, puesto que los puntos que determinan el
* tamaño de la misma cambian. Pero hasta ese momento la expresión
* es una función trigonométrica sencilla cuyo máximo y mínimo
* puede calcularse  fácilmente de forma analítica. El truco es repetir
* el cálculo para todoslos intervalos que encontremos en los primeros
* 90 grados de giro (puesto que la función de área es periódica con 90
* grados, eso es suficiente)
*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <math.h>
#include <vector>
#include <algorithm>

#define MAXAREA   10E20      /* 10E20 > 2^32*2^32 */

static const double PI=3.14159265358979323846;
static const double EPSILON=10E-11;     /* 10E-11 <
arcsen(1/2^32) */

double
min(double a,double b,double c,double d) {
  return min(min(a,b),min(c,d));
}

double
max(double a,double b,double c,double d) {
  return max(max(a,b),max(c,d));
}
//
// Ajusta un ángulo para que esté comprendido entre 0 y 2*PI.
//
void
adjustangle(double &x) {
  if (x<0) {
    x+=2*PI;
  } else if (x>=2*PI) {
    x-=2*PI;
  }
}

//
// Calcula el ángulo desde el origen a las coordenadas (x,y).
//
double
myatan(double x, double y) {
  double a;
  if (x==0) {
    a=PI/2.0;
  } else {
    a=atan(y/x);
  }
  if (fabs(a)<EPSILON) {
    a=0;
  }
  if (x<0) {
    a+=PI;
  }
  adjustangle(a);
  return a;
}

double
modulo(double x, double y) {
  return sqrt(x*x+y*y);
}

//
// Verifica si angle se encuentra comprendido en el ángulo que va
// desde fromangle a toangle.
//
bool
between(double  fromangle,  double  toangle,  double
angle) {
  if (fromangle<toangle) {
    return fromangle<=angle && angle<=toangle;
  } else if (fromangle==toangle) {
    return angle==fromangle;
  } else {
    return angle>=fromangle || angle<=toangle;
  }
}

bool
lessthan(double a, double b) {
  return a-b<-EPSILON;
}

bool
equal(double a, double b) {
  return fabs(a-b)<=EPSILON;
}

class Point {
public:
  double r,a;
  double x,y;

  Point(double _x, double _y): x(_x),y(_y) {
    r=modulo(_x,_y);
    a=myatan(_x,_y);
  }
  Point() {}
  ~Point() {}

  void computexy() {
    x=r*cos(a);
    y=r*sin(a);
  }

  void rotate(double angle) {
    a+=angle;
    adjustangle(a);
    computexy();
  }

//
// Mínimo ángulo positivo que hay que girar el vector (pq)
// para que sea vertical.
//
  static double vertical(const Point &p, const Point
&q) {
    double x=q.x-p.x;
    double y=q.y-p.y;
    if (x==0) {
      return EPSILON;
    }
    double a=myatan(x,y);
    if (between(3*PI/2,PI/2,a)) {
      a=PI/2-a;
      if (a<0) {
```

```
        a+=2*PI;
      }
    } else {
      a=3*PI/2-a;
    }
    return a+EPSILON;
  }
//
// Idem, pero en horizontal.
//
  static double horizontal(const Point &p, const
Point &q) {
    double x=q.x-p.x;
    double y=q.y-p.y;
    if (y==0) {
      return EPSILON;
    }
    double a=myatan(x,y);
    if (between(0,PI,a)) {
      a=PI-a;
    } else {
      a=2*PI-a;
    }
    return a+EPSILON;
  }

};

class Box {
private:
  Point p[4];

public:

//
// Define una caja mediante los cuatro puntos dados.
//
  Box(Point bottom, Point right, Point top, Point
left) {
    p[0]=bottom;
    p[1]=right;
    p[2]=top;
    p[3]=left;
  }
  ~Box() {}

//
// Calcula las áreas máxima y mínima que la caja alcanza entre los
// ángulos fromangle y toangle. Las fórmulas son trigonometría
// básica. En resumen: «A*cos(x+a)+B*cos(x+b)» se puede transfor-
// mar en «C*cos(x+c)»; Y «C*cos(x+c)*D*sen(x+d)» se puede
// transformar en «K+E*sen(x+e)», que es una función cuyo máximo
// y mínimo absoluto se puede calcular fácilmente. También hay que
// comprobar si el máximo y el mínimo están en el intervalo dado, y,
// si no, se utilizan los valores en los extremos del intervalo.
//
  void areas(double fromangle, double toangle, double
&minarea, double &maxarea) {
    double minangle,maxangle;
    double c0,c1,d0,d1;
    double C,D;
    double gamma,delta;
    c0=p[2].x-p[0].x;
    c1=p[2].y-p[0].y;
    C=modulo(c0,c1);
    gamma=myatan(c0,c1);

    d0=p[1].x-p[3].x;
    d1=p[1].y-p[3].y;
    D=modulo(d0,d1);
    delta=myatan(d0,d1);

    maxangle=(PI/2-gamma-delta)/2;
    minangle=maxangle+PI/2;
    adjustangle(maxangle);
    adjustangle(minangle);
    if (maxangle>PI) {
      maxangle-=PI;
    }
    if (minangle>PI) {
      minangle-=PI;
    }

    if (between(fromangle,toangle,minangle)) {
      minarea=C*D*(-1+sin(gamma-delta))/2;
    } else {
      minarea=min(C*D*
(sin(2*fromangle+gamma+delta)+sin(gamma-delta))/2,
                C*D*
(sin(2*toangle+gamma+delta)+sin(gamma-delta))/2);
    }
    if (between(fromangle,toangle,maxangle)) {
      maxarea=C*D*(1+sin(gamma-delta))/2;
    } else {
      maxarea=max(C*D*(sin(2*fromangle+gamma+delta)
      +sin(gamma-delta))/2,
                C*D*(sin(2*toangle+gamma+delta)
      +sin(gamma-delta))/2);
    }
  }

};

class Polygon {
private:
  vector<Point> p;
  unsigned int bottom,top,left,right;

//
// Encuentra el punto que está más abajo y más a la izquierda en el
// vector dado (para comenzar el convex hull).
//
  static unsigned int bottomleft(const vector<Point>
&v) {
    unsigned int k=0;
    for (unsigned int i=1; i<v.size(); i++) {
      if (v[i].y<v[k].y) {
        k=i;
      } else if (v[i].y==v[k].y && v[i].x<v[k].x) {
        k=i;
      }
    }
    return k;
  }

//
// Las siguientes funciones, respectivamente, encuentran los puntos
// que están más abajo y más a la izquierda, más a la izquierda y más
// arriba, más arriba y más a la derecha, y más a la derecha y más
// abajo. Se utilizan para calcular los puntos del polígono que definen
// la caja en cada momento.
//
  unsigned int bottomleft() {
    unsigned int k=0;
    for (unsigned int i=1; i<p.size(); i++) {
      if (lessthan(p[i].y,p[k].y)) {
        k=i;
    } else if (equal(p[i].y,p[k].y) && p[i].x<p[k].x)
{
        k=i;
      }
    }
    return k;
  }
  unsigned int lefttop() {
    unsigned int k=0;
    for (unsigned int i=1; i<p.size(); i++) {
      if (lessthan(p[i].x,p[k].x)) {
        k=i;
    } else if (equal(p[i].x,p[k].x) && p[i].y>p[k].y)
{
        k=i;
      }
    }
    return k;
  }
  unsigned int topright() {
    unsigned int k=0;
    for (unsigned int i=1; i<p.size(); i++) {
      if (lessthan(p[k].y,p[i].y)) {
        k=i;
    } else if (equal(p[k].y,p[i].y) && p[i].x>p[k].x)
{
        k=i;
      }
    }
    return k;
  }
  unsigned int rightbottom() {
    unsigned int k=0;
    for (unsigned int i=1; i<p.size(); i++) {
      if (lessthan(p[k].x,p[i].x)) {
        k=i;
    } else if (equal(p[k].x,p[i].x) && p[i].y<p[k].y)
{
```

```
        k=i;
      }
    }
    return k;
  }

//
// Encuentra el punto del vector dado que, situándose en el punto from,
// se encuentra más a la izquierda (desde from todos los puntos están
// comprendidos en 180 grados o menos). Con esta función se encuentra
// el siguiente punto a from del convex hull en sentido de las agujas del
// reloj.
//
 static unsigned int clockwisemin(unsigned int from,
const vector<Point> &v) {
    unsigned int k;

    if (from==0) {
      k=1;
    } else {
      k=0;
    }
    for (unsigned int i=0; i<v.size(); i++) {
      if (i==from || i==k) continue;
      if ((v[k].x-v[from].x)*(v[i].y-v[from].y)-
         (v[k].y-v[from].y)*(v[i].x-v[from].x) > 0)
{
        k=i;
      }
    }
    return k;
  }

public:

  //
  // El constructor calcula el convex hull del vector de puntos que se
  // pasa como parámetro; el polígono resultante está definido en el
  // sentido de las agujas del reloj, empezando por el punto que esté
  // más abajo y más a la izquierda. Después encuentra los puntos de
  // la caja inicial.
  //
  Polygon(const vector<Point> &v) {
    unsigned int first,current;

    first=bottomleft(v);
    p.push_back(v[first]);
    current=first;
    while (1) {
      current=clockwisemin(current,v);
      if (current==first) {
        break;
      }
      p.push_back(v[current]);
    }

    bottom=0;
    left=lefttop();
    top=topright();
    right=rightbottom();
  }

  void rotate(double angle) {
    for (unsigned int i=0; i<p.size(); i++) {
      p[i].rotate(angle);
    }
  }

//
// Calcula el ángulo mínimo que habría que girar el polígono en el
// sentido contrario de las agujas del reloj para que algún lado quede
// horizontal o vertical. Puesto que el polígono es convexo y está
// definido en el sentido de las agujas del reloj, si los puntos de
// mínima y máxima coordenadas (x) o (y) (los que están tocando la
// caja) son p[bottom], p[left], p[top] y p[right], entonces el primer
// lado en quedar horizontal o vertical tiene que ser uno de
// (p[bottom], p[bottom+1]), (p[left], p[left+1]), (p[top],
// p[top+1]) y (p[right], p[right+1]). Con un poco de trigonometría
// se calculan los valores de cada ángulo, para después elegir el mínimo.
//
  double nextstop() {
    double bottomnext=(bottom+1)%p.size();
    double bottomangle=Point::horizontal(p[bottom],
    p[bottomnext]);
    double leftnext=(left+1)%p.size();
    double leftangle=Point::vertical(p[left],
     p[leftnext]);
    double topnext=(top+1)%p.size();
    double topangle=Point::horizontal(p[top],
    p[topnext]);
    double rightnext=(right+1)%p.size();
    double rightangle=Point::vertical(p[right],
    p[rightnext]);
    return min(bottomangle,leftangle,
     topangle,rightangle);
  }

//
// Calcula las áreas mínima y máxima en que puede empaquetarse el
// polígono. La explicación del algoritmo se puede encontrar al principio
// del programa.
//
  void areas(double &minarea, double &maxarea) {
    double angle,delta;
    double tmpmin,tmpmax;

    maxarea=0;
    minarea=MAXAREA;
    angle=0;
    while (angle<PI/2) {
      delta=nextstop();
      Box *box=new Box(p[bottom],p[right],
       p[top],p[left]);

      box->areas(0,delta,tmpmin,tmpmax);
      if (tmpmin<minarea) {
        minarea=tmpmin;
      }
      if (tmpmax>maxarea) {
        maxarea=tmpmax;
      }
      angle+=delta;
      rotate(delta);
      bottom=bottomleft();
      left=lefttop();
      top=topright();
      right=rightbottom();

      delete box;
    }
  }

};


int
main (int argc, char *argv[]) {
  int n;
  int seqnum=1;
  double a,b;
  vector<Point> v;
  Polygon *polygon;
  double maxarea=0,minarea=0;

  while (1) {
    if (fscanf(stdin,»%d»,&n)!=1) abort();
    if (n==0) break;
    printf(«Gift %d\n»,seqnum++);

    for (int i=0;i<n;i++) {
     if (fscanf(stdin,»%lf %lf»,&a,&b)!=2) abort();
      v.push_back(Point(a,b));
    }
    polygon=new Polygon(v);
    polygon->areas(minarea,maxarea);

     printf(«Minimum area = %.4f\nMaximum area =
%.4f\n»,minarea,maxarea);

    v.clear();
    delete polygon;

    printf(«\n»);
  }
  exit(0);
}
```