Programar es crear

César Sánchez Sánchez

<cesar.sanchez@stanford.edu>

«Empaquetar tapones»: solución

El enunciado de este problema apareció en el número 149 de Novática (enero-febrero 2001, p. 75). Es el programa H de los planteados en el 24º Concurso Internacional de Programación de ACM (2000)

```
// La solución se basa en la siguiente observación:
// si existe una solución entonces existe una solución
// en la que los tapones están lo más pegados posible
// cada uno a un vértice. Así, sólo es necesario probar
// todas las posibilidades tapón-vértice, teniendo en
// cuenta que el tapón puede estar a su vez en dos
// posiciones. En total: 3*2*1 = 6 colocaciones, con
// 8 posiciones, dando un total de 48 pruebas por entrada.

#include <stdio.h>
#include <assert.h>
#include <math.h>

//
// para comparación entre dobles
//
#define EPSILON 0.0001
#define ISZERO(x) ((fabs(x))<EPSILON)
#define FLOAT_EQ(x,y) ((fabs(x-y))<EPSILON)

//
// algunas clases auxiliares
//
class Point {
public:
  Point() {}
  Point(double a,double b) : x(a),y(b) {}
  double x,y;
};

class Vector {
public:
  Vector() {}
  Vector(double a,double b) : x(a),y(b) {}
  Vector(Point hd,Point tl) {
    x = hd.x - tl.x;
    y = hd.y - tl.y;
  }
  Vector perp () { return Vector(-y,x); }
  void normalize () {
    double n = sqrt(x*x + y*y);
    x /= n;
    y /= n;
  }
  double x,y;
};

class Triangle {
public:
  Triangle() {}
  Point vertex[3];
  Vector side[3];
};

class Tops {
public:
  double rad_up(int topid, int pos) {
    assert(topid>=0 && topid<=2);
    return rad[topid][(pos & 1<<topid)?1:0];
  }
  double rad_down(int topid, int pos) {
    assert(topid>=0 && topid<=2);
    return rad[topid][(pos & 1<<topid)?0:1];
  }
  double max(int t) {return
   rad[t][0]>rad[t][1])?rad[t][0]:rad[t][1]; }
  double rad[3][2];
};

//
// calcula el corte entre dos rectas, sabiendo que se cortan
//
Point
cut(Point p1, Vector v1, Point p2, Vector v2) {

  Point res;

  if (ISZERO(v1.x)) {
    res.x = p1.x;
    res.y = p2.y + (res.x - p2.x) * (v2.y/v2.x);
  } else if (ISZERO(v2.x)) {
    res.x = p2.x;
    res.y = p1.y + (res.x - p1.x) * (v1.y/v1.x);
  } else {
    double A = v1.y / v1.x ;
    double B = v2.y / v2.x ;
    res.x = ( p2.y - p1.y + A * p1.x - B * p2.x )
/ (A-B);
    res.y = p1.y + A * (res.x - p1.x);
  }
  return res;
}

//
// distancia entre dos puntos y distancia punto-recta
//
double
dist(Point & c1, Point & c2) {
  return sqrt((c1.x-c2.x)*(c1.x-c2.x)+
              (c1.y-c2.y)*(c1.y-c2.y));

}

double
dist(Point &p, Point &pr, Vector&vr ) {

  Vector perp = vr.perp ();
  perp.normalize ();
  Point p2 = cut (p,perp,pr,vr);
  return dist(p,p2);
}


//
// variables globales para el triángulo y los tapones
//
Triangle the_triangle;
Tops the_tops;


//
// construye el triángulo dadas las longitudes de los lados
//
void
build_triangle(double l1, double l2, double l3) {

  double x = ( l3*l3 - l2*l2 + l1*l1 ) / ( 2 *
l1);
  double y = sqrt( l3*l3 - x*x);

  Point p0(0.0,0.0);
  Point p1(l1 ,0.0);
```

```
  Point p2(x  ,y  );

  the_triangle.vertex[ 0]  = p0;
  the_triangle.vertex[ 1]  = p1;
  the_triangle.vertex[ 2]  = p2;

  the_triangle.side[ 0]  = Vector(p1,p0);
  the_triangle.side[ 1]  = Vector(p2,p1);
  the_triangle.side[ 2]  = Vector(p0,p2);

}


//
// calcula la posición del centro de un tapón «pegado»
// a un vértice dado
Point
calc_pos(int top, int v_id) {
  Vector d1 = the_triangle.side[ (v_id+2)%3];
  Vector d2 = the_triangle.side[ v_id];

  Vector perp1 = d1.perp ();
  perp1.normalize();
  Vector perp2 = d2.perp ();
  perp2.normalize();

  Point p1 = the_triangle.vertex[ v_id];
  p1.x += (perp1.x * the_tops.max(top));
  p1.y += (perp1.y * the_tops.max(top));

  Point p2 = the_triangle.vertex[ v_id];
  p2.x += (perp2.x * the_tops.max(top));
  p2.y += (perp2.y * the_tops.max(top));

  return cut(p1,d1, p2,d2);

}


//
// comprueba si el tapón centrado en el punto dado está
// dentro del triángulo
//
int
is_inside(int topid, Point & c) {
  double rad = the_tops.max(topid);

  if ((dist(c,the_triangle.vertex[ 0],the_triangle.
     side[ 0])<(rad- EPSILON)) ||
     (dist(c,the_triangle.vertex[ 1],the_triangle.
     side[ 1])<(rad-EPSILON)) ||
     (dist(c,the_triangle.vertex[ 2],the_triangle.
     side[ 2])<(rad-EPSILON))) {
   return 0;
  }
  return 1;
}

//
// intenta poner el tapón top0 en el vértice 0,
// el top1 en el vértice 1 y el top2 en el
// vértice2 de todas las maneras posibles
//
int
try_pos(int top0, int top1, int top2) {
  if (top0==top1 || top0==top2 || top1==top2) {
    return 0;
  }

  Point c0 = calc_pos(top0,0);
  Point c1 = calc_pos(top1,1);
  Point c2 = calc_pos(top2,2);

  double dist0 = dist(c0,c1);
  double dist1 = dist(c1,c2);
  double dist2 = dist(c2,c0);

  if ((!is_inside(top0,c0)) ||
     (!is_inside(top1,c1)) || (!is_inside(top2,c2)))
    { return 0;
  }

  // intenta todas las posibles posturas
  // de los tapones
  for (int pos=0; pos<8; pos++) {
    if ((dist0 >= the_tops.rad_up  (top0,pos) +
        the_tops.rad_up  (top1,pos)) &&
        (dist0 >= the_tops.rad_down(top0,pos) +
        the_tops.rad_down(top1,pos)) &&
        (dist1 >= the_tops.rad_up  (top1,pos) +
        the_tops.rad_up  (top2,pos)) &&
        (dist1 >= the_tops.rad_down(top1,pos) +
        the_tops.rad_down(top2,pos)) &&
        (dist2 >= the_tops.rad_up  (top0,pos) +
        the_tops.rad_up  (top2,pos)) &&
        (dist2 >= the_tops.rad_down(top0,pos) +
        the_tops.rad_down(top2,pos))) {
      return 1;
    }
  }
  return 0;
}

int
main(int argc, char * argv[]) {

  unsigned int l1, l2 ,l3;
  int round = 1;
  int i,j,k;

  while (1) {
    if (scanf(«%d %d %d %lf %lf %lf %lf %lf %lf»,
        &l1, &l2, &l3,
        &the_tops.rad[ 0][ 0], &the_tops.rad[ 0][ 1],
        &the_tops.rad[ 1][ 0], &the_tops.rad[ 1][ 1],
        &the_tops.rad[ 2][ 0],&the_tops.rad[ 2][ 1])
        < 9) {
      exit(0);
    }
    if (l1==0 && l2==0 && l3==0) {
      break;
    }
    for (i=0;i<3;i++) {
      the_tops.rad[ i][ 0]/=2.0;
      the_tops.rad[ i][ 1]/=2.0;
    }

    // construye el triángulo
    build_triangle((double)l1,(double)l2,(double)l3);

    // prueba en todas las posiciones
    if (round>1) {
      printf(«\n»);
    }
    printf(«Triangle number %d:\n»,round);
    for (i=0;i<3;i++) {
      for (j=0;j<3;j++) {
        for (k=0;k<3;k++) {
          if (try_pos(i,j,k)) {
            printf(«All three stoppers will fit in
             the triangular space.\n»);
            goto next;
          }
        }
      }
    }
    printf(«Stoppers will not fit in the triangu-
lar space.\n»);
  next:
    round++;
  }
  return 1;
}
```