Program Promenade

連載

丸い紙吹雪

寺田　実（電気通信大学情報通信工学科）
terada@ice.uec.ac.jp

■問題

今回取り上げる問題は，2002年11月に金沢工業大学で行われたアジア地区予選の問題H，“Viva Confetti”である．問題は
http://www.kitnet.jp/icpc/problems
から入手可能である．

さまざまな大きさの円形に切った紙片（confetti）が床に散らばっているときに，そのうちのいくつが（一部でも）上から見えているかを求めるのが問題である（本稿では，「円」という言葉を円周だけでなくその内部も含めて「円盤」のように用いることにする）．

Confettiというのは辞書を引くと「紙吹雪」という説明がある．その形状が円形であったかどうかは筆者には分からない（江戸時代の歌舞伎では，雪をそれらしく見せるために三角形の紙片を用いたそうだが）．

入力となるデータは，まず円の個数があり，それに続いて各円の中心のx, y座標と半径が，下にあるものから順に1行ずつ並ぶ．出力は見えている円の個数である．図-1にデータの例を示す．

問題に関する制限は以下の通り．

```
3
0 0 1
0 0 4
3 -1 3
```

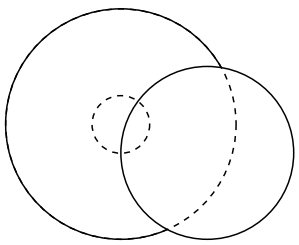

図-1　問題データの例

- 円の数は100以下
- 入力値（浮動小数点数）は小数点以下12桁まで
- 関与する座標値は−10から10の範囲（これが中心位置と半径の上限を与える）

プログラムで使用する変数は以下の通りである：

```
#define MAXDATA 100
int ndata;   /* 円の個数 */
double gx[MAXDATA], /* 中心x座標 */
gy[MAXDATA], /* 中心y座標 */
gr[MAXDATA]; /* 半径 */
```

■準備

この種の幾何学的な問題では，プログラミングの知識だけでなく，幾何の知識も必要になる．この問題の場合は，点の間の距離，円と点，円と円の包含関係，2円の交点などである．

主要な関数は以下の通り：

- 点の間の距離

```
double dist(double x1,double y1,
            double x2,double y2)
```

- 円iが点を含むか－その点と円の中心の距離が円の半径以下

```
int includes_point(int i,
                   double x, double y)
```

- 円iが円jを含むか－（中心間の距離＋円jの半径）が円iの半径以下

```
int includes_circle(int i, int j)
```

ところが，円iと円jの交点を求めるのはそれほど簡単ではない．

円iの中心と交点の距離＝円iの半径　　(1)

円jの中心と交点の距離＝円jの半径　　(2)

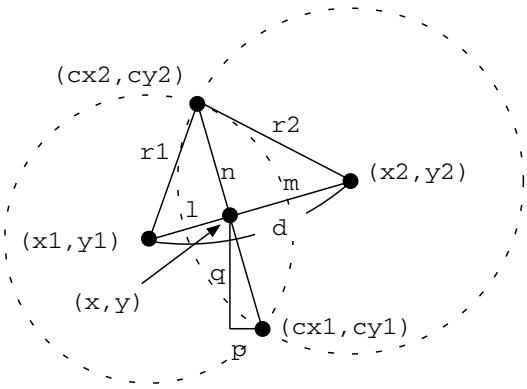

図-2　2円の交点

で式の数は足りているのだが交点の座標を (x, y) として，x, y を解析的に（式の形で）求めようとするとうまくいかない．だがここでは，具体的な数値が与えられたときに x, y が求まればよいので，以下のようにプログラムできる．図-2 にプログラム中に出てくる変数を図示する．式 (1)，(2) と三平方の定理から

$$l^2 + n^2 = r1^2,\ m^2 + n^2 = r2^2$$

中心間の距離を d とすると，$m=d-l$ から，

$$l = (r1^2 - r2^2 + d^2)/2d$$

$$n = \sqrt{r1^2 - l^2}$$

ゆえに

$$x = x1 + (x2-x1)l/d$$

$$y = y1 + (y2-y1)l/d$$

また，三角形の相似から

$$p/n = (y2-y1)/d,\ q/n=(x2-x1)/d$$

であるから，2 つの交点は $(x+p, y-q)$, $(x-p, y+q)$ のように得られる．

```
/* 2円i, jの 交点を求める
   変数引数 (cx1,cy1) (cx2,cy2)
   返り値は交点の数 (0, 1, 2) */
int intersect(int i, int j,
      double *cx1, double *cy1,
      double *cx2, double *cy2)
{
  double x1=gx[i], y1=gy[i], r1=gr[i],
         x2=gx[j], y2=gy[j], r2=gr[j];
  double d = dist(x1, y1, x2, y2);
  double l, n, x, y, p, q;

  if((d > r1+r2) || (d < fabs(r1-r2)))
    return 0;

  l = ((r1*r1 - r2*r2)/d + d) / 2.0;
  n = sqrt(r1*r1 - l*l);

  x = x1 + (x2-x1)*l/d;
  y = y1 + (y2-y1)*l/d;

  p = (y2-y1)*n/d;
  q = (x2-x1)*n/d;

  *cx1 = x+p; *cx2 = x-p;
  *cy1 = y-q; *cy2 = y+q;
  if(d == r1+r2) return 1;
  else return 2;
}
```

また，「ある点を含む最も上位にある円」を求める関数も示しておく．

```
/* 円iから 円j (i<=j) のうち,
   点(x,y)を含む最上位の円を返す
   該当する円がなければ, -1 */
int find_c(double x,double y,int i,int j)
{
  for(; i<=j; j--){
    if(includes_point(j, x, y)){
      return j;
    }
  }
  return -1;
}
```

さて，具体的な解法を検討しよう．

(1) 幾何的な制約から直接求める

円が 2 つであれば，上記の包含判定を用いれば答は直ちに求められる．しかし，3 つ以上になるとそのような単純な方法は思いつかない．1 つの円が複数の円によって隠されるような状況が生じるためである．

(2) 各点でどの円が見えているか

ある座標値が与えられたときに，その位置にどの円が見えているかは，$O(n)$ の手間で簡単に判定できる．したがって，領域すべての点についてこの判定を行えばよいことになるが，それには与えられた精度のもとで 10^{24} 以上の計算が必要で到底望みはない．検討する点の個数を削減できればよいのである．

本稿では，(2) に沿って，検討対象となる点の削減を軸にいくつかの解法を説明していく．

■水平分割法

図 -3 を見ていただきたい．水平に引いた直線 1–6 は，円の交点と，各円の上端と下端を通るものである．2 本の水平線で挟まれた区間に着目すると，その範囲では円弧（と水平線）に囲まれた領域の並び方は定まっている．そこで，その範囲に新たに適当な水平線をとれば，その線上を検討するだけでその範囲全体を検討したことになる．

水平線 3 と 4 の中間に引いた水平線 X がその例である．直線 X 上を検討するには，直線 X と各円の交点を求め，各交点の中間点を検討すれば十分である．図では点 a, b, c の 3 点で見えている円が求まれば，水平線 3, 4 で挟まれた範囲をすべて検討したことになる．

計算の手順は，以下のような 2 重のループとなる：

1. すべての円の交点と，上端下端を求める
2. 1 で求めた点の y 座標をソートする
3. （ループ 1）ソート結果に対して，隣接する y 座標の中間点のそれぞれに対して
   (a) その点を通る水平線と各円との交点の x 座標を求める
   (b) その結果をソートする
   (c)（ループ 2）隣接する x 座標の中間点に対して，そこで見えている円を決定する

この処理の計算量は，円の交点の総数が $O(n^2)$ であるので，外側のループはその回数だけ回る．内側のループは，水平線と円との交点はたかだか $O(n)$ であるからその回数だけ回り，見えている円の判定に要する $O(n)$ を考慮して全体では $O(n^4)$ の手間で処理ができる．

問題の条件は $n \leq 100$ であったから，現在の計算機の能力を考えると問題はないといえる．実際，コンテストの際のテストデータは問題なく処理できた．

さらに，内側のループにおいて，各点で独立に可視の円を判定するのではなく「現在点での円の重なり状況」を工夫して保持すれば判定にかかる手間を $O(\log n)$ に減らすことができ，全体の計算量も $O(n^3 \log n)$ にできる．具体的には，ヒープ構造を用意して，水平線を左から右にスキャンしながら円の開始と終了を記憶していけば，一番上に見えている円が直ちに取り出せる．

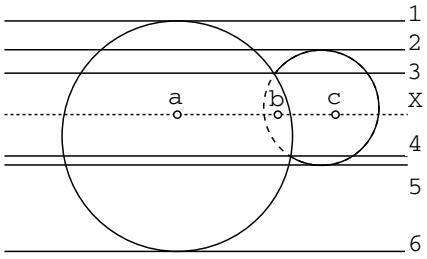

図 -3　水平線による区間分割

なお，このプログラムでは，下請けとして，円 i と水平線 $y = c$ との交点を求める関数

```
double intersect_h(int i, double c)
```

を用いている．この関数の返り値は，交点と中心の x 座標の差である．

```
/* double 同士の大小比較 (qsor t 用 ) */
int cmp_dbl(const void *dp1,
            const void *dp2) {
  /* 定義は省略 */
}
int mid_method(void)
{
  int ny, nx, nc;
  int i, j, k;
  double x1, y1, x2, y2;
  double liney[MAXDATA*(MAXDATA+1)];
  double linex[MAXDATA*2];
  int gvis[MAXDATA];

  ny = 0;
  for(i=0; i<ndata; i++){
    gvis[i] = 0;
    liney[ny++] = gy[i]+gr[i];
    liney[ny++] = gy[i]-gr[i];
    for(j=0; j<i; j++){
      nc = intersect(i,j,&x1,&y1,&x2,&y2);
      if(nc == 2){
        liney[ny++] = y1;
        liney[ny++] = y2;
      } else if(nc == 1){
        liney[ny++] = y1;
      }
   }
  }

  qsort(liney, ny, sizeof liney[0],
  cmp_dbl);
```

```
  for(i=0; i<ny-1; i++){
    y1 = (liney[i] + liney[i+1])/2.0;
    nx = 0;
    for(j=0; j<ndata; j++){
      x1 = intersect_h(j, y1);
      if(x1 >= 0){
        linex[nx++] = gx[j]-x1;
        linex[nx++] = gx[j]+x1;
      }
    }
    qsort(linex, nx, sizeof linex[0],
    cmp_dbl); for(j=0; j<nx-1; j++){
      x1 = (linex[j] + linex[j+1])/2.0;
      k = find_c(x1, y1, 0, ndata-1);
      if(k >= 0) gvis[k]++;
    }
  }

  j = 0;
  for(i=0; i<ndata; i++) if(gvis[i]>0)
  j++; return j;
}
```

■可視交点法

一般に，円の可視領域は，自分自身の円弧か，別の（上にある）円の円弧によって囲まれている．自分自身の円弧だけにより囲まれている場合（つまり円がそのまま完全に見えている場合）を除けば，可視領域を囲む円弧は複数必要で，それらは必ず交点を持つ．

図-4 を見ていただきたい．円1と円2の交点Pの付近を拡大したものである．Pの近くでは，円1と円2に加えて，その点において背景となっている円（ここでは円3）の3つの領域が必ず見えていることが分かる（場合によっては，背景となる円3が存在しない場合もある）．

この知見を使うと，「見えている交点」を調べればそれぞれの交点の近くに見えている円を列挙することができる．つまり，始めに述べた「検討すべき点」を交点だけに限ることになる．

円がそのまま完全に見えている場合には交点が存在しないが，交点を持たない円は完全に隠されるか完全に見えているかのどちらかであるから，円の相互関係から可視かどうかを容易に判定ができる．

計算の手順は以下のようになる：

1. すべての円の交点を求める．
2. それらのうち，「見えている交点」を求める．その条件は，その点において以下の条件を満たすものである：
   (a) 交点をなす上側の円（図でいえば円1）よりも上に円がない
   (b) 上側の円と下側の円との間に別の円が挟まっていない
3. 「見えている交点」の近くの領域の円に印をつける
4. 交点を持たない円については，それより上にある円との包含関係を確認する

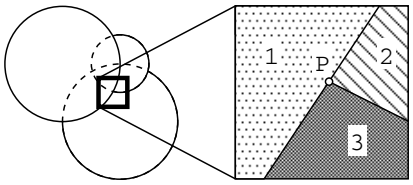

図-4　円の交点

計算量は，交点の総数が $O(n^2)$ であるから，それぞれが見えているかの判定にかかる $O(n)$ をかけて $O(n^3)$ となる．

```
int cross_method(void)
{
  int i, j, k, b, ni, ans=0;
  double x[2], y[2]; /* 2つの交点 */
  int gc[MAXDATA]; /* 自分より上との交点数 */
  int gvis[MAXDATA]; /* 円が見えているか */

  for(i=0; i<ndata; i++){
    gc[i] = 0;
    gvis[i] = 0;
    /* 自分より上の円との交点を求める */
    for(j=i+1; j<ndata; j++){
      ni = intersect(i, j,
             &x[0], &y[0], &x[1], &y[1]);
      gc[i] += ni;
      for(k=0; k<ni; k++)
        if((find_c(x[k],y[k],i+1,j-1)==-1)
          &&(find_c(x[k],y[k],j+1,ndata-1)
             == -1)){
          gvis[i] = 1;
          b = find_c(x[k],y[k],0,i-1);
          if(b != -1)
            /* 背景に円が存在すれば */
            gvis[b] = 1;
        }
    }
  }
```

```
  ans = 0;
  for(i=0; i<ndata; i++)
    if(gvis[i] != 0) ans++;
    else if(gc[i] == 0){
      for(j=i+1; j<ndata; j++)
        if(includes_circle(j, i)) break;
      if(j >= ndata) ans++;
    }
  return ans;
}
```

■分割統治

最後に紹介するのは，平面の再帰的分割をもとにした一種の分割統治法である．これまで紹介した方法が円の交差関係などを利用してきたのに対して，この方法は

- 対象図形が凸であることのみを前提として
- ある点が対象図形に含まれているかの判断ができれば十分

という特徴を持つ．

具体的には，ある円が見えているかどうかを判断するために，平面上の正方形領域に着目し，その中で以下の処理を行う：

1. （再帰の停止条件）正方形が一定以下のサイズになれば不合格
2. まったくその円が見えないことが分かれば不合格
3. 1点でもその円が見えていれば合格
4. そうでない場合，その領域を縦横に4分割し，上記のテストを再帰的に行い，いずれか1つでも合格すれば合格とする

場合2は，正方形領域がその円からはずれているとか，もっと上にある円がその正方形領域すべてを覆う場合が該当する．場合3は，正方形領域の四隅のいずれかにおいてその円が最も上にある場合が該当する．

場合2において，正方形がその円を含まないことの判定には，四隅のすべてがその円に含まれていないことをチェックしなければならない．ところが，円の中心から見て第一象限に位置する正方形だけに話を限ると，その正方形の左下点のチェックだけで十分である．この手法を使って，関数checkは円の四部分のそれぞれについてトップレベル（binary_method）から4回呼び出されている．

```
/* 4点すべて がiより上の円に隠されているか */
int covered(int i, double x0, double x1,
            double y0, double y1)
{
  int j;
  for(j=i+1; j<ndata; j++){
  if(includes_point(j, x0, y0) &&
     includes_point(j, x0, y1) &&
     includes_point(j, x1, y0) &&
     includes_point(j, x1, y1))
    return 1;
  }
  return 0;
}

#define PREC 5.0e-13

int check(int i, double x0, double x1,
          double y0, double y1)
{
  int r;
  double x2, y2;

  if((fabs(x1 - x0) < PREC) ||
     (fabs(y1 - y0) < PREC)) r=0;
  else if(!includes_point(i,x0,y0)) r=0;
  else if(covered(i,x0,x1,y0,y1)) r=0;
  else if((find_c(x0,y0,i,ndata-1)==i)||
          (find_c(x0,y1,i,ndata-1)==i)||
          (find_c(x1,y1,i,ndata-1)==i)||
          (find_c(x1,y0,i,ndata-1)==i))
    r=1;
  else {
    x2 = (x0+x1)/2.0;
    y2 = (y0+y1)/2.0;
    r = check(i, x0, x2, y0, y2) ||
        check(i, x2, x1, y0, y2) ||
        check(i, x0, x2, y2, y1) ||
        check(i, x2, x1, y2, y1);
  }
  return r;
}

int binary_method(void)
{
  int i, n;

  n = 0;
  for(i=0; i<ndata; i++){
    if(check(i, gx[i], gx[i]+gr[i],
                gy[i], gy[i]+gr[i]) ||
       check(i, gx[i], gx[i]-gr[i],
                gy[i], gy[i]+gr[i]) ||
       check(i, gx[i], gx[i]-gr[i],
```

```
                gy[i], gy[i]-gr[i]) ||
        check(i, gx[i], gx[i]+gr[i],
                gy[i], gy[i]-gr[i]))
      n++;
    }

    return n;
  }
```

しかし，残念なことにこの方法ではすべての問題が解けるというわけにはいかない．原理的には大丈夫でも計算時間がオーバーしてしまうケースがあるのである．

この方法では，再帰が停止するためには，正方形領域の中で合格か不合格が決まる必要がある．しかるに，それを決定できる領域（合格の場合ならその円が見えている領域，不合格の場合なら別の円がその円を隠してはみ出している領域）が非常に細かった場合，その大きさにまで正方形を縮小していかなければならない．

図 -5 は，注目する円（点線）がそれより少し大きい円（実線）に隠されて見えない場合の平面分割を示している．はみ出して隠している部分の幅程度の分割が必要になることが見てとれる．その正方形が円周上に並ぶことになるから，全体としての正方形の個数は最小幅の逆数程度が必要となる．

ところが，問題の条件では入力データの精度は 12 桁であるとされており，現実的な時間でこの処理を行うことは困難であろう．

実際，問題文の入力データ例の最後のものは

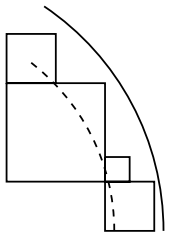

図 -5　分割統治で分割が細かくなる例

```
2
0 0 1
0.0000001 0 1
```

となっており，$x > 0$（右側）の円周がまさにこの状況になっている．このあたりに，出題側のテストデータ作成の工夫が凝らされていると考えられる．

■おわりに

以上述べてきた通り，幾何学的な問題は，プログラミング以外の知識も必要であるし，解法にもバリエーションが多く，難しくも興味深いといえよう．今回は，可視交点法を決定版としたいと思う．

（平成 15 年 5 月 12 日受付）