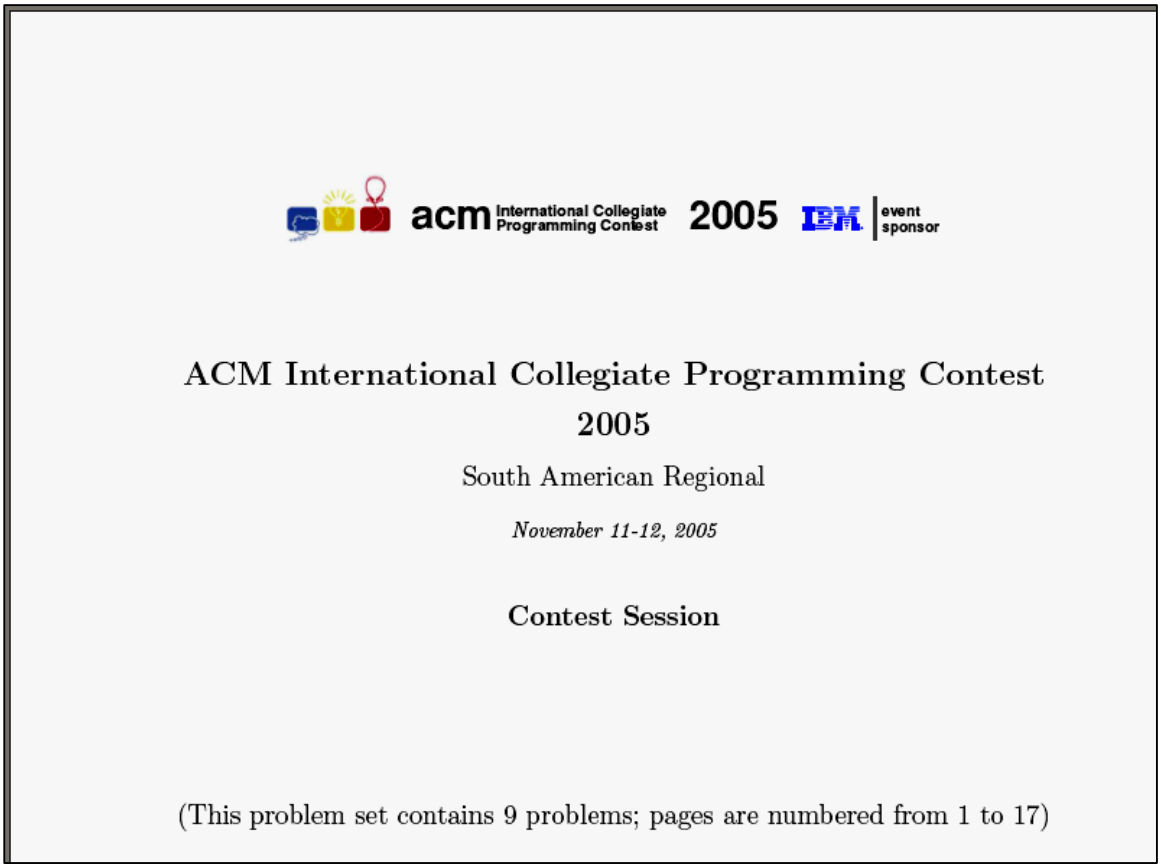


Una Solución (*) del Problema B "Mission Impossible"

Dr. Maynard Kong
(mkong@pucp.edu.pe)

Diciembre de 2005

Preparación de los comentarios por V. Khlebnikov (vkhlebn@pucp.edu.pe)

(*) *Esta solución fue desarrollada sin conocimiento de la solución del autor del problema.*

Comencemos a leer el enunciado:

Problem B
Mission Impossible

Source file name: `mission.c`, `mission.cpp`, `mission.java` *or* `mission.pas`

You have been hired to explore enemy territory. It is risky business, you know that. So, you'd better be prepared! The enemy has placed a number of security points all over his country, from which radars are detecting any moving vehicle within their range of cover. Any such detected object will be immediately destroyed. Fortunately enough, you have been given by your government a map of the enemy territory, consisting of coordinates and radius of coverage of each radar. You have also a list of local informers (together with their locations) that you should contact in order to obtain valuable information. Your mission is to try to contact one of these informers, preferably the one with highest *insider-coefficient*. The insider-coefficient of each informer is simply the distance from the informer to the border of the country, where such a distance is defined as the minimum over all distances from the location of the informer to each point of the border. In intuitive sense, the informer with highest insider-coefficient is that who is located as inside the country as possible, and will presumably have more valuable information about the country.

Your first thought is then to design a computer program which will check if there is a path from your initial location, always the point (2000, 2000), to any of the informers' location, without crossing any region which is covered by radar. Whenever possible, the program should indicate which reachable informer is the one to be contacted, according to the insider-coefficient criteria described above.

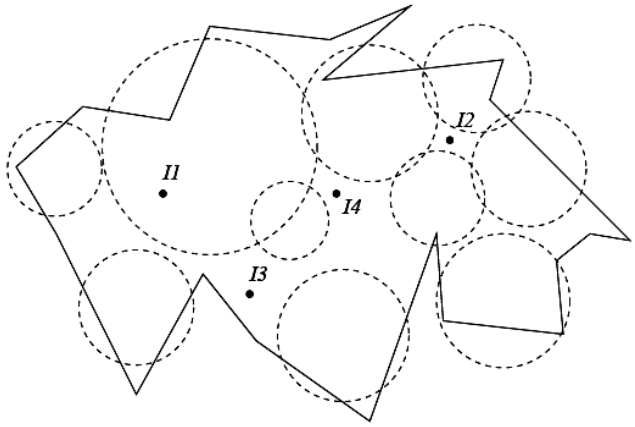


Figure 1: Possible Scenario

Tenemos:

el nombre del archivo de programa:	`mission.c`
el archivo de entrada:	`mission.in`
el archivo de salida:	`mission.out`

Cada radar se describe por sus coordenadas y el radio de alcance, es decir por un círculo. Según la aclaración durante el concurso, el borde de los círculos de los radares está cubierto por los radares.

El valor de cada informante (*the insider-coefficient*) es su distancia mínima de la frontera.

La posición inicial de la ruta es (2000, 2000).

The enemy country has the shape of a *simple polygon* (*not necessarily convex*). Recall that a polygon is called simple if it is described by a single, non-intersecting boundary. The borders of the country will be given as a sequence of X,Y-coordinates corresponding to the sequence of vertexes of the polygon. You may assume that all the radar's centres and the informers' coordinates are located within the country's border. Notice, however, that the area covered by the radars might include regions outside the border.

In the sample scenario of Figure 1, informer $I1$ cannot be contacted since he is inside the region covered by radars. The informer $I2$, although outside the radar's region, can't be contacted either since any trip to his location would go through the deadly radar-covered regions. Both informers $I3$ and $I4$ could be contacted, so that informer $I4$ is chosen since his insider-coefficient is greater than that of $I3$.

No se busca la ruta misma ni su longitud. Sino el hecho de la existencia o ausencia de la ruta a uno de los informantes que tenga el mayor valor (*the insider-coefficient*).

Input

The input consists of several test cases. The first line of each test case describes the border of the enemy country, in the format

```
B X1 Y1 X2 Y2 ... XB YB
```

where $3 \leq B \leq 1000$ is the number of border points, and each Xi Yi is the coordinate of the i-th point in the border. The border of the country consists of line segments between points i and $i+1$, and between points B and 1. The second line gives the number of informers and their respective positions, in the format

```
N X1 Y1 X2 Y2 ... XN YN
```

where $1 \leq N \leq 1000$ is the number of informers, and Xi Yi is the coordinate of the i-th informer. The third line describes the position and radius of the radars, in the format

```
M X1 Y1 R1 X2 Y2 R2 ... XM YM RM
```

where $1 \leq M \leq 30$ is the number of radars, Xi Yi is the coordinate of the i-th radar, and Ri is the radius of the i-th radar. All the coordinates are integers $0 \leq X, Y \leq 1000$. The radius of the radars are integers in the range $1 \leq R \leq 1000$. A test case where $B = N = M = 0$ indicates the end of the input. This test case must not be processed.

The input must be read from standard input.

El ejemplo de entrada y su interpretación:

`4 0 0 0 200 200 200 200 0`	***Caso 1***: $B = 4$ (cantidad de puntos) $(3 \leq B \leq 1000)$, las coordenadas son: (0, 0), (0, 200), (200, 200), (200, 0).
`2 70 70 120 120`	$N = 2$ (cantidad de informantes) $(1 \leq N \leq 1000)$, en las coordenadas (70, 70) y (120, 120).
`1 100 100 100`	$M = 1$ (cantidad de radares) $(1 \leq M \leq 30)$, en las coordenadas (100, 100) con el radio 100.
`4 0 0 0 200 200 200 200 0`	***Caso 2***: $B = 4$ (cantidad de puntos), las coordenadas son: (0, 0), (0, 200), (200, 200), (200, 0).
`3 100 102 70 80 20 10`	$N = 3$ (cantidad de informantes), en las coordenadas (100, 102), (70, 80) y (20, 10).
`4 70 70 35 130 70 35 130 130 35 70 130 35`	$M = 4$ (cantidad de radares), en las coordenadas (70, 70) con el radio 35, en (130, 70) con el radio 35, en (130, 130) con el radio 35 y en (70, 130) con el radio 35.
`0`	$B = 0$ Fin de entrada
`0`	$N = 0$
`0`	$M = 0$

Output

For each test case in the input, your program must produce one line containing either "Mission impossible" or "Contact informer K", where "K" is the index of the informer (as given in the input) with highest insider-coefficient which can be reached by the spy without going inside any radar coverage area. If there are more than one informer satisfying this condition, choose the one among them with lowest index.

The output must be written to standard output.

```
Misión imposible
Contact informer 3
```

Lectura de datos de entrada

Prepararemos el esqueleto del programa y lo probaremos con los datos de prueba:

```
$ cat mision.c
#include <stdio.h>

#define MAX  1001

int XB[MAX];    // puntos de borde
int YB[MAX];
int NB;         // numero de puntos o vertices

int XI[MAX];    // coordenadas de informantes
int YI[MAX];
int NI;         // numero de informantes

int XR[MAX];    // centros de radares
int YR[MAX];
int RR[MAX];    // radios
int NR;         // numero de radares

int
main(void)
{
  int i;

  while (1) {
    scanf("%d", &NB);
    for (i=0; i<NB; i++) scanf("%d %d", XB+i, YB+i);

    scanf("%d", &NI);
    for (i=0; i<NI; i++) scanf("%d %d", XI+i, YI+i);

    scanf("%d", &NR);
    for (i=0; i<NR; i++) scanf("%d %d %d", XR+i, YR+i, RR+i);

    if (NB==0 && NI==0 && NR==0) break;
  }
}
$ gcc mission.c -o mission
$ ./mission <mission.in
$
```

Hasta aquí todo está correcto. El programa ha leído todos los datos.

Procesamiento de los informantes

Se necesita trabajar con cada caso de los informantes. La variable `inf` (con el valor inicial `-1`) indicará el número del informante que nos interesa, pero si ningún informante es alcanzable, su valor inicial se mantendrá:

```
...
int
main(void)
{
  int i;
  int inf;

  while (1) {
    ...
    if (NB==0 && NI==0 && NR==0) break;

    inf=-1;
    for (i=0; i<NI; i++) {

    }

    if (inf==-1) printf("Mission impossible\n");
    else printf("Contact informer %d\n", inf+1);
  }
}
$ gcc mission.c -o mission
$ ./mission <mission.in
Mission impossible
Mission impossible
$
```

Se puede descartar algunos informantes sin calcular sus coeficientes de importancia (*insider-coefficient*) si ellos no están en el área, están en el alcance de radares o están rodeados por los radares.
Así aparece la primera línea del cuerpo de la estructura `for`:

```
    ...
    inf=-1;
    for (i=0; i<NI; i++) {
      if (!admisible(i)) continue;
    }
```

La función `admisible()` será la siguiente:

```
int
admisible(int inf)      // prueba del informante inf
// Punto de informante es admisible
//   si es interior a la region de borde,
//   si no esta dentro del alcance de los radares, y
//   si no es punto encerrado por uno de los ciclos de radares.
{
  int i;

  if ( !interior(XB,YB,NB,XI[inf],YI[inf]) ) return 0;
  if ( DentroRadar(XI[inf],YI[inf]) ) return 0;

  for (i=0; i<nCiclos; i++)
    if ( interior(XC[i],YC[i],tamCiclo[i],XI[inf],YI[inf]) ) return 0;

  return 1;
}
```

Necesitaremos algunas declaraciones adicionales sobre los ciclos de radares:

```
#define MAXC    40
int Ciclo[MAXC][MAX];   // arreglo para registrar "ciclos"
int tamCiclo[MAX];      // formados por centros de radares
int nCiclos;
int XC[MAXC][MAX];      // coordenadas de puntos de ciclos
int YC[MAXC][MAX];

int
interior(int *X, int *Y, int N, int Xinf, int Yinf)
{
  return 0;
}

int
DentroRadar(int Xinf, int Yinf)
{
  return 0;
}
```

El desarrollo de la función `DentroRadar()` es simple:

```
int
DentroRadar(int Xinf, int Yinf)
{
  int i;
  double d;

  for (i=0; i<NR; i++) {
    d=sqrt( (XR[i]-Xinf)*(XR[i]-Xinf)+(YR[i]-Yinf)*( YR[i]-Yinf) );
    if ( d<=RR[i] ) return 1;
  }
  return 0;
}
```

El uso de la función `sqrt()` implica la necesidad de incluir en el programa su encabezado e indicar al compilador el uso de la librería matemática:

```
#include <math.h>

$ gcc mission.c –o mission -lm
$ ./mission <mission.in
Mission impossible
Mission impossible
$
```

Para la función `interior()` se necesitarán dos funciones auxiliares. Una para determinar si la ordenada de un punto pertenece a un segmento, y la otra para obtener la abscisa del punto que pertenece a una recta según su ordenada:

```
int
entreYs(int y1, int y2, int y)
{
  if ( y1<=y && y<=y2 ) return 1;
  if ( y2<=y && y<=y1 ) return 1;
  return 0;
}
```

```
int
obtieneX(int *X, int *Y, int i, int y)
{
  int m;

  m=(X[i+1]-X[i])/(Y[i+1]-Y[i]);
  return m*(y-Y[i])+X[i];
}
```

¿Dentro del polígono?

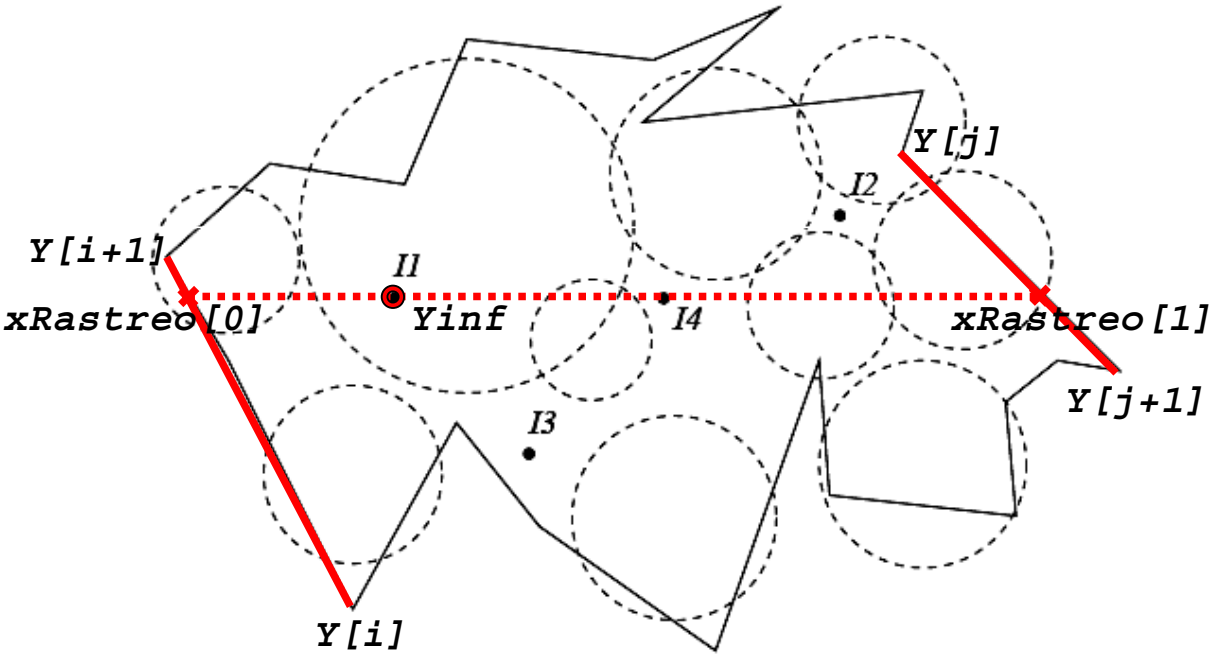
Para obtener el polígono se necesita encerrarlo conectando el último punto con el primer:

```
...
int
main(void)
{
    ...
    if (NB==0 && NI==0 && NR==0) break;

    XB[NB]=XB[0]; YB[NB]=YB[0]; // cerrar borde

    inf=-1;
    ...
  }
}
```

Para determinar si, por ejemplo, el informante *I1* está o no dentro del polígono, necesitamos identificar los bordes que lo limitan. Entonces se debe recorrer todos los bordes y verificar si la ordenada del informante está entre las ordenadas de los extremos del borde.



Esto lo hace el siguiente código (que será la primera parte de la función `interior()`):

```
  // determinar lados cuyas ordenadas contienen Yinf
  for (i=0; i<N; i++)
    if ( entreYs(Y[i],Y[i+1],Yinf) )
      if ( Y[i]==Y[i+1] ) {
        xRastreo[nR++]=X[i]; xRastreo[nR++]=X[i+1];
      } else xRastreo[nR++]=obtieneX(X,Y,I,Yinf);
```

donde el arreglo `xRastreo` contiene las abscisas de líneas de rastreo de la ordenada `Yinf` del informante con los bordes del polígono:

```
  // abscisas de linea de rastreo Yinf con poligonal
  int xRastreo[MAX];
  int nR=0;
```

Así, en la figura presentada, obtenemos los valores `xRastreo[0]` y `xRastreo[1]`.

Ahora se necesita ordenar los puntos de rastreo porque el orden de ingreso de los puntos de bordes es arbitrario. Aquí está la segunda parte de la función `interior()`:

```
  // ordenar xRastreo
  for (i=0; i<nR; i++) {
    posmin=i;
    for (j=i+1; j<nR; j++)
      if ( xRastreo[j]<xRastreo[posmin] ) {
        posmin=j; temp=xRastreo[i];
        xRastreo[i]=xRastreo[posmin]; xRastreo[posmin]=temp;
      }
  }
```

La última parte de la función `interior()` determina si la abscisa `Xinf` del informante está entre los dos puntos `xRastreo` consecutivos, lo que significará que es un punto interior (siempre y cuando el primer punto es par, o sea es el inicio del segmento):

```
  // detector si punto (Xinf,Yinf) esta dentro de region
  for (i=0; i<nR-1; i++)
    // se determina si abscisa Xinf esta entre dos X consecutivos
    // si el indice i es par, el punto es interior (i empieza de 0)
    if ( xRastreo[i]<=Xinf && Xinf<=xRastreo[i+1] ) {
      if ( i%2 == 0 ) resultado=1; break;
    }
  return resultado;
}
```

El código completo de la función `interior()` es el siguiente:

```
int
interior(int *X, int *Y, int N, int Xinf, int Yinf)
{
  int i,j;

  // abscisas de linea de rastreo Yinf con poligonal
  int xRastreo[MAX];
  int nR=0;

  int posmin, temp, resultado=0;
```

```
  // determinar lados cuyas ordenadas contienen Yinf
  for (i=0; i<N; i++)
    if ( entreYs(Y[i],Y[i+1],Yinf) )
      if ( Y[i]==Y[i+1] ) {
        xRastreo[nR++]=X[i]; xRastreo[nR++]=X[i+1];
      } else xRastreo[nR++]=obtieneX(X,Y,I,Yinf);

  // ordenar xRastreo
  for (i=0; i<nR; i++) {
    posmin=i;
    for (j=i+1; j<nR; j++)
      if ( xRastreo[j]<xRastreo[posmin] ) {
        posmin=j; temp=xRastreo[i];
        xRastreo[i]=xRastreo[posmin]; xRastreo[posmin]=temp;
      }
  }

  // detector si punto (Xinf,Yinf) esta dentro de region
  for (i=0; i<nR-1; i++)
    // se determina si abscisa Xinf esta entre dos X consecutivos
    // si el indice i es par, el punto es interior (i empieza de 0)
    if ( xRastreo[i]<=Xinf && Xinf<=xRastreo[i+1] ) {
      if ( i%2 == 0 ) resultado=1; break;
    }
  return resultado;
}
```

Ciclos de radares

Para saber si un informante está encerrado en un círculo formado por los radares, se necesita identificar si las áreas de alcance de algunos radares forman un círculo. Se añade a la función principal la invocación a la función `creaCiclos()` que se desarrolla:

```
...
int
main(void)
{
    ...
    if (NB==0 && NI==0 && NR==0) break;

    XB[NB]=XB[0]; YB[NB]=YB[0]; // cerrar borde
    creaCiclos();
    inf=-1;
    ...
  }
}

...
void
hallaCiclo(int radar)
{
}

void
creaCiclos()
{
  int i;

  nCiclos=0;
  for (i=0; i<NR; i++) hallaCiclo(i);
}
```

Tomamos los radares uno por uno y buscamos el ciclo. Cuando avanzamos con cada radar puede ser que este radar ya está en un ciclo construido. Por eso necesitaremos la función auxiliar `estaEnCiclo()`:

```
int
estaEnCiclo(int radar)
{
  int i,j;

  for (i=0; i<nCiclos; i++)
    for (j=0; j<tamCiclo[i]; j++)
      if ( radar==Ciclo[i][j] ) return 1;
  return 0;
}
```

Ahora se puede construir los ciclos:

```
void
hallaCiclo(int radar)
{
  int CicloTemp[MAX];
  int nCentro, centro_ant, centro_sig, i;
  int agregado;

  CicloTemp[0]=radar; nCentro=1;
```
El radar dado será el primer centro del ciclo posible.
```
  // para simplificar se asumen ciclos disjuntos ...
  if ( estaEnCiclo(radar) ) return;
```
El radar dado ya está incluido en un ciclo, salimos de la función.
```
  agregado=1;
```
Se espera que podamos agregar más radares al ciclo.
```
  while (1) {
    if ( !agregado ) break;
```
Si ya no se puede agregar, abandonamos el bucle.
```
    centro_ant=CicloTemp[nCentro-1];
```
Partimos del último radar del ciclo.
```
    agregado=0;
```
No podremos agregar el siguiente radar antes de verificarlo.
```
    for (centro_sig=radar+1; centro_sig<NR; centro_sig++) {
```
Recorremos todos los radares siguientes.
```
      for (i=0; i<nCentro; i++)
        if ( centro_sig==CicloTemp[i] ) break;
      if ( i<nCentro ) continue;
```
Si el siguiente radar ya está en el ciclo, el bucle se interrumpe antes de completarse y, en este caso, pasar al siguiente radar.
```
      if ( centrosVecinos(centro_ant,centro_sig) ) {
        agregado=1; CicloTemp[nCentro++]=centro_sig;
        centro_ant=centro_sig;
      }
```
Si las áreas de estos dos radares tienen una intersección, entonces el radar analizado debe ser incluido en el ciclo.
```
    }
  }
```
Aquí se terminó la construcción del ciclo de radares.
```
  if ( nCentro>=3 ) {
```
Para formar un ciclo se necesitan, por lo menos, 3 radares.

Falta verificar solamente que el área del último radar en la cadena tiene una intersección con el área del primer radar de la cadena.
```
    centro_ant=CicloTemp[nCentro-1];
    centro_sig=CicloTemp[0];
    if ( centrosVecinos(centro_ant,centro_sig) ) {
```
Registrar el ciclo analizado como un ciclo válido.

```
      for (i=0; i<nCentro; i++) Ciclo[nCiclos][i]=CicloTemp[i];
      tamCiclo[nCiclos]=nCentro;
      nCiclos++;
    }
  }
}
```

Se usaron las siguientes funciones auxiliares:

```
// distancia entre dos puntos
double
distPuntos(int x1, int y1, int x2, int y2)
{
  return sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1));
}

// las areas de dos radares tienen una interseccion?
int
centrosVecinos(int c1, int c2)
{
  if ( distPuntos(XR[c1],YR[c1],XR[c2],YR[c2]) <= (RR[c1]+RR[c2]) )
    return 1;
  return 0;
}
```

En el arreglo `Ciclo[][]`, el primer índice corresponde al número del ciclo y el segundo índice corresponde al número del radar que forma parte de este ciclo. Entonces la segunda parte de la función `creaCiclos()` prepara las coordenadas de estos radares en los arreglos `XC[][]` y `YC[][]`:

```
void
creaCiclos()
{
  int i,j;

  nCiclos=0;
  for (i=0; i<NR; i++) hallaCiclo(i);

  for (i=0; i<nCiclos; i++) {
    for (j=0; j<tamCiclo[i]; j++) {
      XC[i][j]=XR[Ciclo[i][j]]; YC[i][j]=YR[Ciclo[i][j]];
    }
    XC[i][j]=XC[i][0]; YC[i][j]=YC[i][0];
  }
}
```

Lo único que falta es completar la función principal:

```
int
main(void)
{
  int i;
  int inf;

  while (1) {
    scanf("%d", &NB);
    for (i=0; i<NB; i++) scanf("%d %d", XB+i, YB+i);

    scanf("%d", &NI);
    for (i=0; i<NI; i++) scanf("%d %d", XI+i, YI+i);

    scanf("%d", &NR);
```

```
    for (i=0; i<NR; i++) scanf("%d %d %d", XR+i, YR+i, RR+i);

    if (NB==0 && NI==0 && NR==0) break;

    XB[NB]=XB[0]; YB[NB]=YB[0]; // cerrar borde
    creaCiclos();
    inf=-1;
    for (i=0; i<NI; i++) {
      if ( !admisible(i) ) continue;
      distTEMP=distanciaBorde(XI[i],YI[i]);
      if ( inf==-1 ) {
        inf=i; distMAX=distTEMP;
      } else
        if ( distTEMP>distMAX ) {
          inf=i; distMAX=distTEMP;
        }
    }

    if (inf==-1) printf("Mission impossible\n");
    else printf("Contact informer %d\n", inf+1);
  }
```

Código completo

El código completo presentado contiene algunas modificaciones con los comentarios correspondientes:

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

#define MAX  1001

int XB[MAX];      // puntos de borde
int YB[MAX];
int NB;           // numero de puntos, o vertices

int XI[MAX];      // coordenadas de informantes
int YI[MAX];
int NI;           // numero de informantes

int XR[MAX];      // centros de radares
int YR[MAX];
int RR[MAX];      // radios
int NR;           // numero de radares

#define MAXC      40
int Ciclo[MAXC][MAX];   // arreglo para registrar "ciclos"
int tamCiclo[MAX];      // formados por centros de radares
int nCiclos;
int XC[MAXC][MAX];      // coordenadas de puntos de ciclos
int YC[MAXC][MAX];

// ------------------------------------------------------------------------

int
entreYs(int y1, int y2, int y)
{
  if ( y1<=y && y<=y2 ) return 1;
  if ( y2<=y && y<=y1 ) return 1;
  return 0;
}
```

```
// ------------------------------------------------------------------

int
obtieneX(int *X, int *Y, int i, int y)
{
  int m;

  m=(X[i+1]-X[i])/(Y[i+1]-Y[i]);
  return m*(y-Y[i])+X[i];
}

// ------------------------------------------------------------------

int
interior(int *X, int *Y, int N, int Xinf, int Yinf)
{
  int i,j,k;

  // abscisas de linea de rastreo Yinf con poligonal
  int xRastreo[MAX];
  int nR=0;

  int posmin, temp, resultado=0;

  // determinar lados cuyas ordenadas contienen Yinf
  for (i=0; i<N; i++)
    if ( entreYs(Y[i],Y[i+1],Yinf) )
      if ( Y[i]==Y[i+1] ) {
        xRastreo[nR++]=X[i]; xRastreo[nR++]=X[i+1];
      } else xRastreo[nR++]=obtieneX(X,Y,i,Yinf);

  // ordenar xRastreo
  for (i=0; i<nR; i++) {
    posmin=i;
    for (j=i+1; j<nR; j++)
      if ( xRastreo[j]<xRastreo[posmin] ) {
        posmin=j; temp=xRastreo[i];
        xRastreo[i]=xRastreo[posmin]; xRastreo[posmin]=temp;
      }
  }
```

Si Yinf coincide con un extremo de segmento que también es un extremo del siguiente segmento, tendremos dos puntos xRastreo idénticos. Por eso el siguiente código elimina los puntos de segmentos duplicados.

```
  for (i=0; i<nR; i++) {      // eliminar los duplicados
    for (j=i+1; j<nR; j++)
      if ( xRastreo[i]==xRastreo[j] ) {
        for (k=j+1; k<nR; k++) xRastreo[k-1]=xRastreo[k];
        nR--;
      }
  }

  // detectar si punto (Xinf,Yinf) esta dentro de region
  for (i=0; i<nR-1; i++)
    // se determina si abscisa Xinf esta entre dos X consecutivos
    // si el indice i es par, el punto es interior (i empieza de 0)
    if ( xRastreo[i]<=Xinf && Xinf<=xRastreo[i+1] ) {
      if ( i%2 == 0 ) resultado=1; break;
    }
  return resultado;
}

// ------------------------------------------------------------------
```

```
int
DentroRadar(int Xinf, int Yinf)
{
  int i;
  double d;

  for (i=0; i<NR; i++) {
    d=sqrt( (XR[i]-Xinf)*(XR[i]-Xinf)+(YR[i]-Yinf)*(YR[i]-Yinf) );
    if ( d<=RR[i] ) return 1;
  }
  return 0;
}

// ------------------------------------------------------------------

int
estaEnCiclo(int radar)
{
  int i,j;

  for (i=0; i<nCiclos; i++)
    for (j=0; j<tamCiclo[i]; j++)
      if ( radar==Ciclo[i][j] ) return 1;
  return 0;
}

// ------------------------------------------------------------------

double
distPuntos(int x1, int y1, int x2, int y2)
{
  return sqrt((x2-x1)*(x2-x1)+(y2-y1)*(y2-y1));
}

// ------------------------------------------------------------------

int
centrosVecinos(int c1, int c2)
{
  if ( distPuntos(XR[c1],YR[c1],XR[c2],YR[c2]) <= (RR[c1]+RR[c2]) )
    return 1;
  return 0;
}

// ------------------------------------------------------------------

void
hallaCiclo(int radar)
{
  int CicloTemp[MAX];
  int nCentro, centro_ant, centro_sig, i;
  int agregado;

  CicloTemp[0]=radar; nCentro=1;
  // para simplificar se asumen ciclos disjuntos ...
  if ( estaEnCiclo(radar) ) return;
  agregado=1;
  while (1) {
    if ( !agregado ) break; // no se encontro ningun vecino
    centro_ant=CicloTemp[nCentro-1]; // de este radar partimos
    agregado=0;
    for (centro_sig=NR-1; centro_sig>radar; centro_sig--) {
```

```
      for (i=0; i<nCentro; i++)
        if ( centro_sig==CicloTemp[i] ) break;
      if ( i<nCentro ) continue;    // este ya esta en el ciclo

      if ( centrosVecinos(centro_ant,centro_sig) ) {
        agregado=1; CicloTemp[nCentro++]=centro_sig;
        centro_ant=centro_sig;
      }
    } // for sig
  }   // while (1)

  if ( nCentro>=3 ) {
    centro_ant=CicloTemp[nCentro-1];
    centro_sig=CicloTemp[0];
    if ( centrosVecinos(centro_ant,centro_sig) ) {
      for (i=0; i<nCentro; i++) Ciclo[nCiclos][i]=CicloTemp[i];
      tamCiclo[nCiclos]=nCentro;
      nCiclos++;
    }
  }
}

// ------------------------------------------------------------------

void
creaCiclos(void)
{
  int i,j;

  nCiclos=0;
  for (i=0; i<NR; i++) hallaCiclo(i);

  for (i=0; i<nCiclos; i++) {
    for (j=0; j<tamCiclo[i]; j++) {
      XC[i][j]=XR[Ciclo[i][j]]; YC[i][j]=YR[Ciclo[i][j]];
    }
    XC[i][j]=XC[i][0]; YC[i][j]=YC[i][0];
  }
}

// ------------------------------------------------------------------

int
admisible(int inf)      // prueba del informante inf
// Punto de informante es admisible
//   si es interior a la region de borde,
//   si no esta dentro del alcance de los radares, y
//   si no es punto encerrado por uno de los ciclos de radares.
{
  int i;

  if ( !interior(XB,YB,NB,XI[inf],YI[inf]) ) return 0;

  if ( DentroRadar(XI[inf],YI[inf]) ) return 0;

  for (i=0; i<nCiclos; i++)
    if ( interior(XC[i],YC[i],tamCiclo[i],XI[inf],YI[inf]) ) return 0;

  return 1;
}

// ------------------------------------------------------------------
```

```
double
distanciaLado(int i, int x, int y)
{
  double lado, lado1, lado2, s;

  lado=distPuntos(XB[i],YB[i],XB[i+1],YB[i+1]);
  lado1=distPuntos(XB[i],YB[i],x,y);
  lado2=distPuntos(XB[i+1],YB[i+1],x,y);
  s=(x-XB[i]) * (YB[i+1]-YB[i]) - (y-YB[i]) * (XB[i+1]-XB[i]);
  if ( lado1>lado ) return lado2;
  if ( lado2>lado ) return lado1;
  s=(s>0)?s:(-s); // s es area del paralelogramo
  return (s/lado);      // lado es la base, h=s/lado=altura
}

// ------------------------------------------------------------------

double
distanciaBorde(int x, int y)
{
  int i;
  double dmin, d;

  for (i=0; i<NB; i++) {
    d=distanciaLado(i,x,y);
    if ( i==0 || d<dmin) dmin=d;
  }
  return dmin;
}

// ------------------------------------------------------------------

int
main(void)
{
  int i,j,x,y,r,r_found,r_uniq;
  int inf;
  double distTEMP, distMAX;

  while(1) {

    scanf("%d", &NB);
    for (i=0; i<NB; i++) scanf("%d %d", XB+i, YB+i);

    scanf("%d", &NI);
    for (i=0; i<NI; i++) scanf("%d %d", XI+i, YI+i);

    scanf("%d", &NR);
```

En el archivo de datos de entrada de jueces se encuentran los radares repetidos. Por eso el siguiente código los elimina.

```
    for (i=0; i<NR; i++) {
      scanf("%d %d %d", &x, &y, &r);
      if ( i==0 ) {
        XR[0]=x; YR[0]=y; RR[0]=r; r_uniq=1;
      } else {
        r_found=0; j=0;
        while ( !r_found && j<r_uniq ) {
          if ( x==XR[j] && y==YR[j] && r==RR[j] ) r_found=1;
          j++;
        }
        if ( !r_found ) {
          XR[r_uniq]=x; YR[r_uniq]=y; RR[r_uniq]=r;
          r_uniq++;
        }
```

```
      }
    }
    if ( NR != 0 ) NR=r_uniq;

    if (NB==0 && NI==0 && NR==0) break;

    XB[NB]=XB[0]; YB[NB]=YB[0];     // cerrar borde
    creaCiclos();
    inf=-1;
    for (i=0; i<NI; i++) {
      if ( !admisible(i) ) continue;
      distTEMP=distanciaBorde(XI[i],YI[i]);
      if ( inf==-1 || distTEMP>distMAX ) {
        inf=i; distMAX=distTEMP;
      }
    }

    if (inf==-1) printf("Mission impossible\n");
    else printf("Contact informer %d\n", inf+1);
  }
}
```

```
$ gcc mission.c -o mission -lm
$ ./mission <mission.in
Contact informer 1
Contact informer 1
Mission impossible
Mission impossible
Contact informer 2
Contact informer 5
Contact informer 10
Contact informer 7
Contact informer 1
Contact informer 1
Contact informer 2
Contact informer 1
Contact informer 1
Mission impossible
Contact informer 1
Contact informer 1
Contact informer 1
Contact informer 1
Contact informer 340
Contact informer 574
Contact informer 100
Mission impossible
$ ./mission <mission.in >mission.out
$ cmp mission.out mission.sol
$
```

Algunas observaciones

1. El problema puede ser considerado de extrema dificultad para el concurso.
2. La repetición de radares en el archivo de datos de entrada podría considerarse como un error.
3. En la solución presentada se asume que los ciclos de los radares son disjuntos.