*Originalmente publicado en los números 156 (enunciado) y 157 (solución) de la revista Novática (http://www.ati.es/novatica/)*

# Problema E: Almejas gigantes e interfaces de usuario

A. Herranz
aherranz@fi.upm.es

P. Sánchez
psanchez@skyrealms.org

M. Carro
mcarro@fi.upm.es

J. Mariño
jmarino@fi.upm.es

## 1. Descripción del problema

Durante varios años, los investigadores de la Asociación para la Conservación del Mar han estado entrenando varias formas de vida marina con el objeto de construir un ordenador subacuático que pueda verse desde el espacio exterior. La investigación se ocupa actualmente en una clase de almeja gigante, de nombre científico *Panope abrupta*. Dichas almejas pueden pesar hasta cuatro kilogramos y medir hasta un metro con sus sifones (o cuellos) totalmente extendidos. Dada su esperanza de vida (hasta 150 años) prometen ser buenos agentes para implementar un interfaz de usuario oceánico a gran escala.

El trabajo actual se centra en pares de almejas entrenadas situadas en esquinas diferentes de una rejilla rectangular. Las almejas se desplazan en la rejilla desprendiendo un producto químico luminiscente que está contenido en recipientes adheridos a su concha. Las almejas están entrenadas para moverse una unidad de longitud en la rejilla en dirección horizontal o vertical para aproximar un vector director (cada almeja tiene un único vector). Si un movimiento lleva a una almeja fuera del espacio delimitado por la rejilla, un delfín entrenado la transporta inmediatamente a la celda situada en la parte opuesta de la rejilla. El punto de reentrada en la rejilla está alineado horizontal o verticalmente con la celda de salida y la trayectoria de la almeja se mantiene. Los movimientos de las almejas están sincronizados.

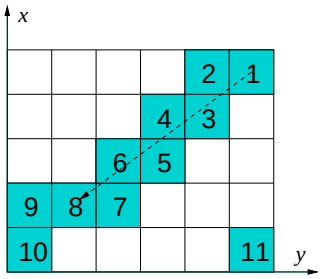
Una almeja se detiene al entrar en una celda que ya ha sido visitada. Si dos almejas intentan entrar en la misma celda al mismo tiempo, se detienen en esa celda. Si dos almejas intentan entrar cada una en una celda ocupada por la otra a la vez, ambas se detienen. Las almejas están inicialmente situadas en casillas de las esquinas de modo que su vector director apunte hacia el interior de la rejilla: es decir, si el componente $x$ es positivo y el componente $y$ es negativo, la posición inicial estará en el valor mínimo de la $x$ y en máximo de la $y$ en la rejilla.

Ambas almejas empiezan en el instante $t = 1$ en sus respectivas (y distintas) esquinas. Una almeja sigue su vector como si el *punto de anclaje* del mismo estuviese en el centro de la celda inicial de la almeja en la rejilla. Siempre se mueve a la siguiente celda dividida en regiones por el vector (o su extensión) con una excepción: si el vector atraviesa exactamente un punto de corte de las líneas que constituyen la rejilla, la almeja se mueve primero horizontalmente y después verticalmente para llegar a la siguiente celda dividida por su vector.

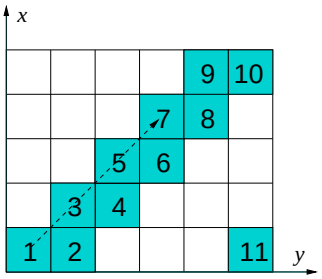
La figura 1 muestra varios caminos realizados por almejas. Los números en cada celda representan el tiempo transcurrido. Las celdas están numeradas desde el punto más abajo y a la izquierda empezando en cero en las direcciones $x$ e $y$. Si las dos almejas de la figura 1 empiezan al mismo tiempo en la misma rejilla de $5 \times 6$ celdas, ambas se detendrán tras cinco unidades de tiempo con un total de 10 celdas iluminadas.

Debes escribir un programa para seleccionar pares de almejas que iluminen el máximo número de celdas de la rejilla en el menor tiempo. Los cálculos deben repetirse para varios tamaños de rejillas y combinaciones de almejas, que constituirán los diferentes casos de prueba.

### Descripción de la entrada




Una almeja con vector (-4, -3) que se mueve en una rejilla de 6 × 5 empezando en la celda (5, 4). Se detiene en el paso 12 porque regresa a (5, 4).




Una almeja con vector (1, 1) que se mueve en una rejilla de 6 × 5 empezando en la celda (0, 0). Se detiene en el paso 12 porque pasa de nuevo por (0, 0).

Figura 1: Caminos de la almeja

La entrada consiste en una secuencia de casos de prueba, cada uno encabezado con una línea que contiene dos enteros $m$ y $n$, $1 \leq m, n \leq 50$, en los que $m$ y $n$ no son simultáneamente 1. $m$ y $n$ son, respectivamente, las dimensiones $x$ e $y$ de la rejilla. La segunda línea de cada caso de prueba contiene un entero $k, 2 \leq k \leq 10$, que representa el número de almejas. Al menos un par de almejas tienen distintos puntos de arranque. Las siguientes $k$ líneas contienen cada una un par de enteros no nulos que representan los componentes $x$ e $y$ del vector director de la $k$-ésima almeja.

El último caso de prueba está seguido por un par de ceros.

## Descripción de la salida

Hay que imprimir, para cada caso de prueba, el número máximo de celdas iluminadas, el número mínimo de unidades de tiempo requeridas para iluminar esas celdas y los números, dentro de la secuencia inicial de la entrada, de los pares de almejas que logran esos valores. Hay que imprimir todos los pares de almejas que realizan el máximo de iluminación en el tiempo mínimo. El orden en que se realiza la impresión no importa; sin embargo, no se debe imprimir ningún par dos veces para el mismo caso de prueba. El formato de la salida debe ser similar al del ejemplo mostrado más abajo.

## Ejemplo de entrada

```
6 5
3
-4 -3
1 1
1 -1
0 0
```

## Salida para el ejemplo de entrada

```
Caso 1    Celdas pintadas: 10   Tiempo mínimo: 5
  Identificadores de almejas: 1 2
  Identificadores de almejas: 1 3
```

## 2. Introducción

Recordemos que el problema del interfaz gráfico basado en almejas planteaba la creación de un ordenador biológico con una pantalla de dimensiones gigantescas cuyos pixels eran iluminados por unas enormes almejas que obedecían ciertas reglas estrictas (y no triviales) de movimiento. Los experimentos iniciales requerían hallar ciertos parámetros de comportamiento de almejas.

En particular se necesitaba estudiar un conjunto de almejas, cada una moviéndose en una dirección prefijada y arrancando de una esquina en dirección al centro, y determinar el par de almejas que hubiesen iluminado más celdas hasta su detención. Las almejas se mueven en una retícula rectangular a la que se da un topología toroidal (al salir por un borde se entra por el borde contrario). Sin entrar en detalles, ya expuestos en el número anterior, la detención se produce cuando las almejas entran en una celda que ya ha sido visitada por ella misma o por su pareja de recorrido. Se requiere devolver el tiempo invertido y el número de celdas visitadas por el par de almejas que iluminan más celdas en menos tiempo.

A pesar de que la solución propuesta no es especialmente complicada, existen numerosos detalles en el movimiento y la detección de la parada que deben ser tenidos en cuenta. La estrategia elegida ha sido la de realizar una simulación de movimientos de las almejas hasta su detención y escoger el mejor par. Una decisión importante, que hace el código más claro, ha sido la de realizar el cálculo del recorrido de cada almeja separadamente del cálculo de las celdas iluminadas. Esto aísla detalles en cada una de las secciones y, adicionalmente, evita recalcular el recorrido realizado por cada almeja. El cálculo de las celdas visitadas por cada almeja en solitario es, hasta cierto punto, especulativo, ya que se detiene cuando la almeja se encuentra con su propio rastro (lo cual constituye el recorrido más largo que esa almeja podrá realizar).

## 3. Detección del cruce de dos almejas

La detección de los cruces de dos almejas es, en sí mismo, un problema truculento. Veámoslo suponiendo que hemos calculado de antemano las trayectorias máximas (en forma de secuencias de coordenadas indexadas por el tiempo en que llegan a la celda correspondiente) que pueden seguir dos almejas, $A$ y $B$:

$$\begin{array}{ll} A: & a_1, a_2, a_3, a_4, a_5, a_6 \ldots a_k \\ B: & b_1, b_2, b_3, b_4, b_5, b_6 \ldots b_l \end{array}$$

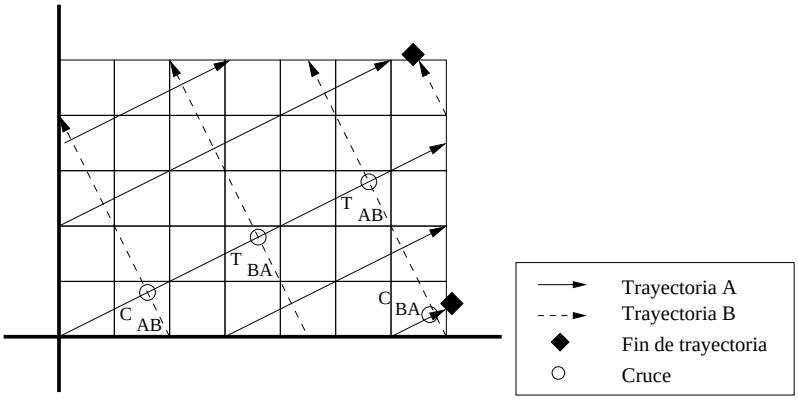
Una almeja se detiene cuando su siguiente movimiento la lleva a una celda por la que ha pasado otra almeja o ella misma, o cuando ambas llegan a la vez a la misma celda. Por tanto, una condición necesaria de detención de la almeja $A$ en el rastro de la almeja $B$ es

$$\exists i \exists j (a_i = b_j)$$

Puede haber varios cruces en las trayectorias de las almejas, pero de ellos sólo es interesante el que detiene a cada almeja. Podemos definir este primer cruce como:

$$C_{AB} = \text{mín}\{i \in 1 \ldots k | \exists j \in 1 \ldots l.\ a_i = b_j\}$$

Sin embargo es posible que este cruce sea *falso*: en la figura




vemos que $C_{AB}$, el corte con el menor recorrido de $A$, no es significativo, porque $B$ no habría llegado nunca a ese punto. Es necesario asegurar que la almeja que choca llegue al corte en un momento posterior (o igual) al de la almeja que dejó el rastro. Por tanto, el primer corte real de $A$ con el trazo de $B$, $T_{AB}$, se puede definir como:

$$T_{AB} = \text{mín}\{i \in 1 \ldots k | \exists j \in 1 \ldots l.\ a_i = b_j \wedge j \leq i\}$$

La figura anterior muestra un caso extremo en el que el camino de cada almeja cubre casi la totalidad de la retícula antes de cruzarse consigo misma.

Para la discusión que viene a continuación es interesante referirse al instante en que $B$ pasó por ese primer punto de corte de $A$ con $B$. Lo llamaremos $P_{BA}$:

$$b_{P_{BA}} = a_{T_{AB}}$$

Aún hay otra circunstancia que se debe tener en cuenta. En el ejemplo de la figura, $B$ impacta con el segmento del trazo de $A$ comprendido entre el origen y $a_{T_{AB}}$. ¿Qué habría pasado si el impacto se hubiese producido en una casilla *posterior* a $T_{AB}$? Tal choque no sería real, pues la almeja $A$ se detuvo ya. Llamaremos *choque fantasma* a una situación así.

Para tener en cuenta esa situación definimos una nueva magnitud que sí tiene en cuenta la dependencia mutua entre las dos almejas y que llamaremos $R$. Podemos distinguir dos casos: si $T_{AB} \leq T_{BA}$, entonces $R_{AB} = T_{AB}$. Si, por el contrario, $T_{AB} > T_{BA}$ nos aseguraremos de que $A$ sólo pueda impactar con su propio trazo o con el segmento inicial del trazo de $B$:

$$\begin{aligned} R_{AB} = \text{mín}(k, \text{mín}\{i \in 1 \ldots k |\ \exists j \in 1 \ldots l\ . \\ (a_i = b_j \wedge j \leq i \wedge j \leq P_{BA})\}) \end{aligned}$$

A partir de los tiempos en que se produce el corte puede deducirse la cantidad de celdas iluminadas. Podemos distinguir dos casos: si $R_{AB} = R_{BA} \wedge A_{R_{AB}} = B_{R_{BA}}$, el número de casillas iluminadas $L$ será la suma de las celdas iluminadas por cada almeja, $R_{AB} - 1$ y $R_{BA} - 1$, más la celda en la que se encuentran:

$$L = (R_{AB} - 1) + (R_{BA} - 1) + 1 = R_{AB} + R_{BA} - 1$$

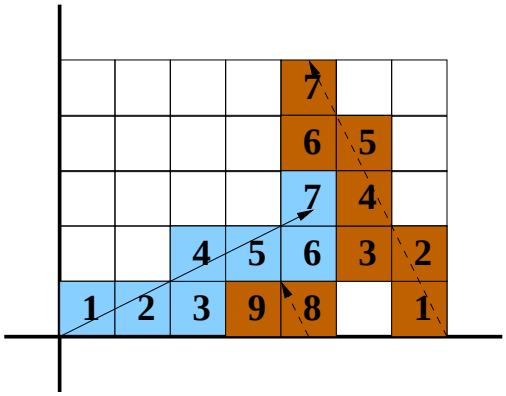
En otro caso, será:

$$L = (R_{AB} - 1) + (R_{BA} - 1) = R_{AB} + R_{BA} - 2$$

En el ejemplo anterior tendríamos

$$L = R_{AB} + R_{BA} - 2 = 8 + 10 - 2 = 16$$

como puede apreciarse en la figura:




Codificar directamente las fórmulas anteriores puede llevar a serias ineficiencias, pues los caminos máximos tienen una longitud, en el peor caso, proporcional al *área de la rejilla* y las fórmulas del párrafo anterior introducen a su vez un factor cuadrático sobre dicha longitud.

La alternativa es simular el avance del par de almejas, con lo cual se detectan los cruces con una complejidad menor, al tenerse en cuenta la ordenación temporal de las casillas recorridas por cada almeja. Lamentablemente, la solución inmediata consistente en hacer una simulación por cada par de almejas tiene un problema grave de *recomputación*, pues al final se calcularían (parcialmente) los caminos de cada almeja $n^2$ veces.

Este problema se evita realizando una única simulación con todas las almejas avanzando por la rejilla, considerando los cruces que se vayan produciendo, como si se realizase una *inundación*. Esta solución es más engorrosa de programar por la cantidad de datos que hay que manejar a la vez. Para hacerla más digerible hemos optado por una solución híbrida en la que los caminos máximos de cada almeja se precomputan mientras que la detección de los cruces se calcula mediante inundación.

Al tratarse de dos problemas independientes entre sí podrían ser abordados, en un concurso, por dos participantes diferentes:

- Cálculo del recorrido que realizará una almeja antes de detenerse (cuando se cruce con su propio recorrido). Un sencillo problema geométrico con sus propios detalles.
- Simulación de los cruces de las almejas, tomando como dato los recorridos de las mismas, previamente calculados.

# 4. Implementación

La solución al problema ha sido implementada en Ada 95. Comenzaremos mostrando la distintas estructuras de datos elegidas para codificar la información que utiliza el algoritmo que presentaremos en la sección 4.2. El precómputo de los recorridos se presenta en la sección 4.3, para terminar en la sección 4.4 con una descripción del cómputo de celdas iluminadas y tiempos.

## 4.1. Estructuras de datos

El principal objetivo del simulador es calcular una tabla con los instantes en los que cada almeja se cruza con el recorrido de otra. Para ello se ha diseñado la siguiente estructura de datos en la que se mantiene actualizada una tabla que indica que en un determinado instante (`Inst_Actual`) en qué instante anterior una almeja se cruzó con el recorrido de otra (`Inst_De_Cruce`):

```
type Simulador is record
   N_Filas, N_Columnas : Positive;
   N_Almjs : Identificador_Almj;
   Almj : Secuencia_De_Almjs;
```

```
   Inst_Actual : Tiempo;
   Inst_De_Cruce :
      Tabla_De_Insts_De_Cruce;
end record;
```

Los tipos `Tiempo`, `Fila` y `Columna` son tipos *subrango* apropiadamente escogidos y `Secuencia_De_Almjs` se han definido así:

```
type Secuencia_De_Almjs is
   array (Identificador_Almj)
      of Informacion_Almj;
```

La información asociada a cada almeja queda representada por el tipo abstracto de datos `Informacion_Almj`. Dicha información se construye a partir de su vector de dirección y del tamaño la rejilla tal como mostraremos en la sección 4.3 y puede accederse a través de las siguientes operaciones:

```
-- Devuelve el último instante antes de
-- que la almeja se cruce con su propio
-- recorrido.
function Inst_Maximo
  (A : Informacion_Almj)
   return Tiempo;

-- PRE: I in 1 .. Inst_Maximo (A)
-- Devuelve la posición de la almeja
-- en un determinado instante
function Poscn_En_Inst
  (A : Informacion_Almj; I : Tiempo)
   return Poscn;

-- PRE: I > 0
-- Decide si la almeja ha parado en el
-- instante indicado
function Ha_Parado
  (A: Informacion_Almj; I: Tiempo)
   return Boolean;

-- PRE: Pos.Y < S.N_Filas /\
--      Pos.X < S.N_Columnas
-- Devuelve el instante en el que la
-- almeja visitó la posición. Devuelve 0
-- si la almeja no visitó dicha posición.
function Inst_De_Visita
  (A: Informacion_Almj; Pos: Poscn)
   return Tiempo;
```

Estas operaciones son en mayor o menor medida *observadores* triviales de la estructura de datos `Informacion_Almj`:

```
type Recorrido is
   array (Tiempo range 1 .. Tiempo'Last)
      of Poscn;
type Tabla_Insts_De_Visita is
    array (Fila, Columna) of Tiempo;
```

```
type Informacion_Almj is record
   N_Filas, N_Columnas : Positive;
   V : Vector;
   Inst_Maximo : Tiempo;
   Recorrido_Maximo : Recorrido;
   Inst_De_Visita :
      Tabla_Insts_De_Visita;
end record;
```

Los instantes de cruce de cada almeja con sus compañeras se guardan en una tabla definida por:

```
type Tabla_De_Insts_De_Cruce is
   array (Identificador_Almj,
          Identificador_Almj)
      of Tiempo;
```

que recoge (sección 4.2) los momentos de cruce de las almejas.

## 4.2. Algoritmo

Antes de comenzar con la descripción del algoritmo recordemos la semántica y las restricciones de uso de los datos de un simulador $s$ del tipo `Simulador`:

- $s$`.Almj (`$a$`)`: Información de la almeja $a$, $a \leq s$`.N_Almjs`
- $s$`.Inst_Actual`: Instante para el que los datos calculados en $s$`.Inst_De_Cruce` son significativos.
- $s$`.Inst_De_Cruce (`$a$`, `$b$`)`: Instante en el que la almeja $a$ se cruzó con el recorrido de la almeja $b$, o 0 si todavía no se ha producido el cruce. Debe cumplirse que $a \leq s$`.N_Almjs` y $b \leq s$`.N_Almjs`.

El algoritmo utilizado para resolver el problema consiste en iniciar una simulación del movimiento de todas las almejas simultáneamente anotando en la tabla `Inst_De_Cruce` el instante en que una almeja se cruza con el recorrido de otra. La simulación se verá simplificada por la potencia del tipo `Informacion_Almj` al tener precomputado el recorrido de cada almeja así como los instantes en los que una almeja visita una posición de manera que el código de la simulación no se mezcla con el problema de calcular la siguiente posición de una almeja.

Al margen del precómputo de los recorridos máximos de las almejas, el algoritmo hace avanzar el tiempo y en cada nuevo instante se realiza, *grosso modo*, el siguiente trabajo:

- Por cada almeja $a$, si no ha finalizado ya su recorrido máximo:
  1. Se obtiene la siguiente celda a la que se desplaza dicha almeja $a$.
  2. Por cada almeja $b$:
     *a*) Se comprueba si la posición ya ha sido visitada por $b$ y, si es así, si lo hizo antes de cruzarse con $a$. En este caso se anota el instante anterior como instante en el que $a$ se cruza con $b$.
     *b*) Si no, se comprueba si $a$ y $b$ están coincidiendo sobre dicha posición en este preciso instante. En este caso se anota el instante como instante en el que $a$ se cruza con $b$.

El algoritmo termina cuando todas las almejas han finalizado sus recorridos.

El grueso del trabajo realizado por el algoritmo ha sido implementado en una operación que toma un simulador `S` y, para el instante `Inst = S.Inst_Actual + 1` y por cada almeja `A`, se ejecuta el siguiente código:

```
if not Ha_Parado(S.Almj(A), Inst) then

 Pos_A:= Poscn_En_Inst(S.Almj(A), Inst);

 for B in 1 .. S.N_Almjs loop
  if -- No empiezan en la misma esquina
     Poscn_En_Inst(S.Almj(A), 1)
       /= Poscn_En_Inst(S.Almj(B), 1) then
   if not Se_Ha_Cruzado(S, A, B) then
    if --  B pasó por Pos_A
       Ha_Pasado(S, B, Pos_A) and then
       --  y lo hizo antes de cruzarse con A
       (not Se_Ha_Cruzado(S, B, A) or else
        Inst_De_Visita(S.Almj(B), Pos_A)
          <= S.Inst_De_Cruce(B, A)) then

     S.Inst_De_Cruce(A, B) := Inst - 1;

    elsif --  B no se ha cruzado con A
          not Se_Ha_Cruzado (S, B, A) and
          --  y coinciden en Instante
          Coinciden
           (S.Almj(A), S.Almj(B), InstB) then

     S.Inst_De_Cruce(A, B) := Inst;

    end if; --  B pasó...
   end if; -- not Se_Ha_Cruzado(S, A, B)
  end if; --  No empiezan en la misma esquina
 end loop;
end if; --  not Ha_Parado(S.Almj(A), Inst)
```

donde las operaciones auxiliares `Se_Ha_Cruzado`, `Ha_Pasado`, `Ha_Parado` y `Coinciden` son triviales.

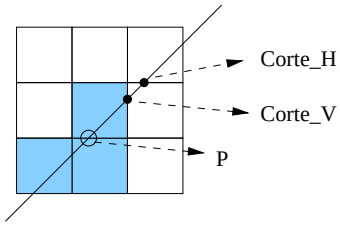
## 4.3. Precomputando los recorridos de las almejas

El trabajo de computar la siguiente celda a la que ha de dirigirse una almeja es sencillo aunque algo laborioso. La decisión de precomputar todo el recorrido de cada almeja tiene como objetivo principal no contaminar el algoritmo de la simulación con los cálculos necesarios para determinar la siguiente celda a visitar.

El primer paso será determinar la celda inicial del recorrido para lo cual el enunciado del problema proporciona como único dato que las almejas comienzan en una esquina con su vector de dirección orientado hacia el interior de la rejilla. Esto determina de manera unívoca la celda inicial salvo en el caso de que el vector sea paralelo a alguno de los ejes de coordenadas, situación que no puede ocurrir porque el enunciado establece explícitamente que las componentes del vector son enteros no nulos.

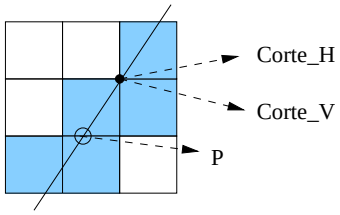
Hay que tener en cuenta que, aunque el resultado de los recorridos de las almejas pertenecerá al plano discreto (celdas), los cálculos de intersecciones se están realizando utilizando números de coma flotante, debido a la necesidad de calcular los cruces con las divisiones de la rejilla. Así mismo, las posiciones deberán estar dentro de los límites de la rejilla de topología toroidal para lo cual se realiza un reposicionamiento de la celda calculada cuando ésta sale fuera de los límites de la rejilla.

Una vez claro esto, el proceso toma el punto anterior de corte con las rectas de la rejilla (`P`) y el vector director de la almeja (`V`) y obtiene los puntos de corte con las dos rectas (`Corte_H` y `Corte_V`) que definen el límite de la celda en la dirección de avance como se muestra en la siguiente figura:




La intersección más cercana, atendiendo a la distancia euclídea, determinará la celda adyacente a la que se avanza.

Un caso especial aparece cuando ambos puntos de corte coinciden, es decir, la almeja salta de celda justo por la esquina. Recordemos que en dicho caso el enunciado especificaba que primero se pasaría a la celda adyacente horizontalmente iluminando en segundo lugar aquélla con la que la original compartía la esquina. Esto puede apreciarse detalladamente en la figura




que hace patente que en este paso se incluyeron dos celdas en el recorrido de la almeja. Por lo tanto, en el algoritmo principal de simulación no será necesario considerar este hecho quedando más homogéneo.

El siguiente fragmento muestra la creación de la información de una almeja incluyendo el cálculo de la secuencia de celdas seguida por la almeja, tal como se acaba de describir:

```
procedure Crear
  (N_Filas    : in     Positive;
   N_Columnas : in     Positive;
   V          : in     Vector;
   A          :    out Informacion_Almj)
is
   Pos : Poscn;   -- Siguiente posición
   -- Siguiente punto de intersección
   Inters : Punto;
   -- Posición intermedia si la intersec-
   -- ción se produce por el centro
   Pos_Inter : Poscn;
begin
 -- Inicializando la información básica
 A.N_Filas := N_Filas;
 A.N_Columnas := N_Columnas;
 A.V := V;
 A.Inst_De_Visita :=
    (others => (others => 0));

 -- Calculando la posición de partida
 Pos.X :=
  (N_Columnas - 1) * Boolean'Pos(V.X >= 0);
 Pos.X :=
  (N_Filas - 1) * Boolean'Pos(V.X >= 0);
```

```
 A.Inst_Maximo := 1;
 A.Recorrido_Maximo (A.Inst_Maximo) :=
    Pos;
 A.Inst_De_Visita (Pos.X, Pos.Y) :=
    A.Inst_Maximo;

 -- Calculando la primera intersección
 Inters :=
   Sig_Inters
    ((X => Float (Pos.X) + 0.5,
      Y => Float (Pos.Y) + 0.5),
      V);
 while True loop
  Pos := Sig_Poscn (A, Pos, Inters);

  if Es_Entero (Inters.X) and
     Es_Entero (Inters.Y) then
   -- Cruza por la esquina
   Pos_Inter :=
      A.Recorrido_Maximo (A.Inst_Maximo);
   Pos_Inter.X :=
      (Pos_Inter.X + Signo (V.X))
         mod N_Columnas;

   if A.Inst_De_Visita
         (Pos_Inter.X,
          Pos_Inter.Y) = 0 then
    A.Inst_Maximo := A.Inst_Maximo + 1;
    A.Inst_De_Visita
       (Pos_Inter.X,
        Pos_Inter.Y) := A.Inst_Maximo;
    A.Recorrido_Maximo (A.Inst_Maximo) :=
       Pos_Inter;
   else
    exit;
   end if;
  end if;

  if A.Inst_De_Visita
        (Pos.X, Pos.Y) = 0 then
   A.Inst_Maximo := A.Inst_Maximo + 1;
   A.Inst_De_Visita (Pos.X, Pos.Y) :=
      A.Inst_Maximo;
   A.Recorrido_Maximo (A.Inst_Maximo) :=
      Pos;
  else
   exit;
  end if;

  Inters:= Sig_Inters (Inters, V);
 end loop;
end Crear;
```

Aunque las operaciones `Sig_Inters` y `Sig_Poscn` son engorrosas consideramos importante

mostrarlas para que el lector observe los múltiples detalles a los que hay que prestar atención. Se habrá observado que las comprobaciones de cruce con las intersecciones de la rejilla se realizan mediante una llamada a `Es_Entero`, que debe tener en cuenta posibles errores de redondeo. Incluso esto no es suficiente como se puede apreciar en el primer comentario de la operación `Sig_Inters`.

```
function Sig_Inters
          (P : Punto;
           V : Vector) return Punto
is
 Corte_H : Punto;
 Corte_V : Punto;
begin
 -- Supongamos que P.X = 1 en la realidad
 -- ‘‘real’’ del problema. Por culpa de
 -- la precisión es fácil que P.X fuera
 -- internamente 0.999999. ¿Cuál sería
 -- el resultado de
 -- Corte_V.X = Float’Floor (P.X) + 1.0?
 if V.X > 0 then
  Corte_V.X :=
    Float’Floor (P.X + EPSILON) + 1.0;
 else
  Corte_V.X :=
    Float’Ceiling (P.X - EPSILON) - 1.0;
 end if;
 Corte_V.Y := P.Y +
   (Corte_V.X * Float(V.Y)/Float(V.X)) -
   (P.X * Float(V.Y)/Float(V.X));

 if V.Y > 0 then
  Corte_H.Y :=
    Float’Floor (P.Y + EPSILON) + 1.0;
 else
  Corte_H.Y :=
    Float’Ceiling (P.Y - EPSILON) - 1.0;
 end if;
 Corte_H.X := P.X +
    (Corte_H.Y * Float(V.X)/Float(V.Y)) -
    (P.Y * Float(V.X)/Float(V.Y));

 --  El resultado es el corte más cercano
 if Distancia (P, Corte_H) <
    Distancia (P, Corte_V) then
  return Corte_H;
 else
  return Corte_V;
 end if;
end Sig_Inters;


function Sig_Poscn
  (A    : Informacion_Almj;
 Pos_Prev   : Poscn;
```

```
 Inters : Punto) return Poscn
is
 Sig_Pos : Poscn := Pos_Prev;
begin
 if --  Intersección con la vertical
    Es_Entero (Inters.X) then
  Sig_Pos.X :=
    (Sig_Pos.X + Signo (A.V.X))
       mod A.N_Columnas;
 end if;

 if --  Intersección con la horizontal
    Es_Entero (Inters.Y) then
  Sig_Pos.Y :=
    (Sig_Pos.Y + Signo (A.V.Y))
       mod A.N_Filas;
 end if;
 return Sig_Pos;
end Sig_Poscn;
```

Obsérvese, además, cómo el cálculo de los cortes con las coordenadas $x$ e $y$ intenta evitar errores de redondeo multiplicando/dividiendo números grandes y sumando/restando números pequeños.

### 4.4. Determinando el mejor par de almejas

El último paso es el más sencillo de todos los descritos y básicamente consiste en la búsqueda de un máximo: por cada par de almejas, aquél que maximice el número de celdas iluminadas y en caso de empate, aquel par que utilice el mínimo tiempo para iluminarlas. El cálculo de celdas iluminadas está realmente en la tabla `Inst_De_Cruce` una vez terminada la simulación.

Finalmente, la salida entrega como información las celdas iluminadas y el tiempo mínimo, pero además deberá volver a calcularse el número de celdas iluminadas y el tiempo para cada par de almejas puesto que deben ofrecerse todos los pares con dicho resultado.

## 5. Últimos pensamientos

**Algoritmo *tonto*** En principio, el algoritmo híbrido (caminos máximos precomputados/detección de cruces simulada) es necesario para evitar la ineficiencia de codificar directamente las fórmulas de la sección 3. Puede tener sentido estudiar la viabilidad de una solución más o menos directa. Los cálculos de intersecciones de secuencias de posiciones no tienen por qué ser cuadráticos si se realiza, por ejemplo, una ordenación lexicográfica, lo cual reduciría la complejidad a lineo-logarítmica.

**Racionales vs. coma flotante** El uso directo de números de coma flotante para resulver problemas cuyo resultado debe ser discreto puede necesitar numerosos artificios. Una clara posibilidad para evitar esto es el uso de paquetes de números reales exactos o, lo que es suficiente en nuestro caso, de operaciones con números racionales.

**¿Técnicas gráficas?** El recorrido de una almeja tiene bastante en común con los algoritmos de dibujo de rectas usados en técnicas gráficas (e.g., el algoritmo de Bresenham). Queda para el lector ver cómo sería de fácil adaptar ese algoritmo (u otro similar) para el caso que nos ocupa, y resolver las manías de las almejas al cruzar una intersección, teniendo en cuenta además que el vector director debe arrancar del centro de una celda.

## 6. Agradecimientos

Querríamos agradecer la oportunidad que la revista Novática nos ha dado de poder contribuir con esta serie de artículos sobre los problemas de la final mundial del concurso de programación de la ACM. Querríamos reconocer especialmente la incansable energía y la paciencia sin fin de Cristóbal Pareja, y, junto con Alberto Verdejo, sus comentarios siempre constructivos acerca de las soluciones de las que nos hemos ido haciendo cargo. También querríamos expresar nuestra gratitud al equipo editorial de Novática, en particular a Rafael Fernández Calvo.