CENTRO DE INVESTIGACION Y DE ESTUDIOS AVANZADOS DEL

INSTITUTO POLITECNICO NACIONAL

DEPARTAMENTO DE INGENIERIA ELECTRICA

SECCION DE COMPUTACION

"UN ALGORITMO PARA ENRUTAMIENTO NO LOCAL"

Tesis que presenta el Ing. Diego Alfonso Andrade Stacey para obtener el grado de MAESTRO EN CIENCIAS en la especialidad de INGENIERIA ELECTRICA con opción de COMPUTACION. Trabajo dirigido por el Dr. Renato Barrera Rivera.

Becario de UNESCO

México, D.F.,27 de Noviembre de 1987.

AGRADECIMIENTO

El autor agradece a la Sección de Computación del Departamento de Ingeniería Eléctrica del CINVESTAV en la persona de su Director el Dr. Manuel E. Guzmán R.. A los profesores de la Sección, en especial al Dr. Renato Barrera Rivera por sus invalorables enseñanzas y asesoría para una exitosa culminación de estos estudios de Maestría; al Dr. Josef Kolar por su dedicación y sugerencias en la revisión de la Tesis. A UNESCO y en especial al Dr. Juan Carlos Anselmi por la ayuda que brinda a los estudiantes no mexicanos para que puedan venir a México a efectuar estudios de postgrado. A los amigos y compañeros de la Sección de Computación un agradecimiento muy sincero por su permanente apoyo y amistad para que esta estadía en México haya sido tan positiva y provechosa.

A Lucre,

Juan Esteban,

y María Isabel

A Laurita,

a la memoria de Alfonso

CONTENIDO

INTRODUCCION.

El problema de determinar la ruta o trayectoria de longitud mínima entre dos puntos de un plano referenciado por un sistema de ejes coordenados (x,y) ha sido tratado extensamente en diversas áreas tales como la investigacion de operaciones, diseño de circuitos integrados [1],[2],[4], movimiento de robots [3],[5]; entre otras aplicaciones.

Un aspecto especial y muy utilizado sobre todo en el diseño de circuitos y en robótica es aquel de obtener la ruta mas corta entre dos puntos en presencia de obstáculos o los cuales pueden corresponder sea a poligonos cerrados o a segmentos rectilineos.

La solución tradicional que se ha dado a la obtención de rutas de longitud mínima esta enmarcada en la teoría de gráficas, con algoritmos como el de Dijsktra.

En el ámbito de la determinacion de la ruta más corta en presencia de obstáculos en los métodos de enrutamiento locales los vecinos de un punto son algunos de los puntos de una figura geométrica que rodea al punto.

Algunos de estos métodos locales y los consiguientes algoritmos que de ellos se desprenden idealizan al plano como una malla o retícula [2], dividiendo al plano en intervalos de longitud unitaria en cada dirección. Los puntos de intersección de las líneas trazadas por estos intervalos corresponden a los vértices de una gráfica.

En este tipo de modelo la ruta se genera pasando de punto a punto de la retícula y tomando en cuenta como entorno de un punto a los puntos adyacentes a ese vértice, exclusivamente.

En contraposición otros métodos consideran un enfoque global del plano para la obtención de la ruta, es decir los vecinos de un punto corresponden a toda una área.

Se presenta en este trabajo de tesis un método para obtener la ruta mas corta entre dos puntos en presencia de obstáculos rectilineos, el mismo que está basado en la partición del plano de estudio en un conjunto de figuras geométricas (cuadriláteros y triángulos) Estas figuras se forman conforme se introducen los obstáculos, de uno en uno.

Para obtener una gráfica con el método e esta tesis, cada figura es un nodo de una gráfica no dirigida. Entre dos nodos hay un lado siempre y cuando las figuras correspondientes sean adyacentes y no exista un obstáculo entre ellas.

El algoritmo de enrutamiento que se describe en este trabajo consiste en efectuar el recorrido de la gráfica de figuras desde la que contiene al punto inicial hasta aquella en la que se ubica el punto final. El recorrido de la gráfica se lleva a cabo con el

concepto de propagación de ranuras, es decir, pasar de un lado de una figura hacia otro lado que sirve de comunicación con una figura adyacente. Los conceptos de la gráfica de figuras y de la propagación de ranuras son ideas originales de este trabajo.

Finalmente, el método descrito en este trabajo permite obtener una ruta de longitud mínima entre dos puntos tanto con la norma de medida l_1 como con la norma de medida l_∞.

Si m es el número de segmentos rectilíneos (obstáculos), el número de figuras geométricas (n) en las que se descompone el plano es generalmente del orden de $n = 3m + 1$ y en el peor de los casos $n = 4m + 1$. Los algoritmos de construcción de la gráfica de figuras y del proceso de enrutamiento son en esperanza de $O(n\log(n))$ pero pueden llegar a ser de $O(n^2)$.

Respecto a la organización del trabajo, en el capítulo 1 se revisan algunos conceptos importantes de teoría de gráficas y de normas de medida, en el capítulo 2 se describen el método y los algoritmos para la construcción de la gráfica de figuras, en el capítulo 3 se plantean el método y algoritmo de enrutamiento, en el capítulo 4 se hace la descripción de otros métodos de enrutamiento y en especial de uno que sirve de comparación con el algoritmo propuesto en esta tesis, en el capítulo 5 se presentan varios ejemplos. Finalmente se presentan algunas conclusiones y comentarios al trabajo.

Los programas que se desarrollaron para esta tesis estan organizados de la siguiente manera:

a) uno para la construcción de la gráfica de figuras, el mismo que tiene aproximadamente 6000 líneas de código fuente.

b) uno para la obtención de la ruta mínima utilizando el método descrito en esta tesis, con alrededor de 1200 líneas de código fuente.

c) uno para la obtención de la ruta mínima utilizando un método comparativo local, con aproximadamente 900 líneas de código fuente.

Todos los programas estan redactados en lenguaje C (compilador Lattice C, versión 3.0).

CAPITULO 1

PRELIMINARES.

En este capitulo se revisan algunos conceptos fundamentales relacionados con Teoria de Gràficas y normas de medidas. Estos conceptos serân frecuèntemente utilizados en los restantes capitulos de este trabajo.

1.1. GRAFICAS.

1.1.1. CONCEPTOS FUNDAMENTALES.

Una gràfica es una estructura de datos definida como:

G = [V,E] donde

V = conjunto de vèrtices de la gràfica
E = conjunto de lados de la gràfica.

La gràfica puede ser dirigida en cuyo caso todo lado es un par ordenado de vèrtices distintos, o no dirigida en cuyo caso todo lado es un par no ordenado de vèrtices distintos. En el presente trabajo no se consideran ciclos (es decir lados de la forma (v,v)).

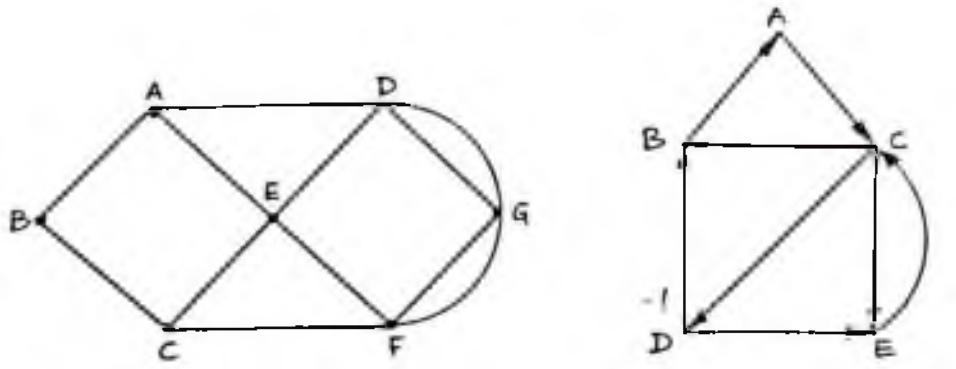
Si (v,w) es un lado no dirigido de la gràfica v y w son vèrtices adyacentes y constituyen los extremos del lado. Nòtese que se usan los parèntesis curvos para simbolizar lados no dirigidos. En un lado dirigido, [v,w], el sentido de este lado es desde v hasta w y se utilizarán los parèntesis rectangulares para simbolizar los lados dirigidos.

Si (v,w) es cualquier lado, (v,w) es incidente a v y w y v y w son incidentes a (v,w). En general, si S es un conjunto de vèrtices un lado es incidente a S si uno de sus extremos pertenece a S.

Si v es un vèrtice de una gràfica no dirigida su grado es el nùmero de vèrtices adyacentes.

Si v es un vèrtice de una gràfica dirigida su grado interior es el nùmero de lados [u,v] (que llegan a v) y su grado exterior es el nùmero de aristas [v,w] (que salen de v).

En la Fig. 1.1 se presentan ejemplos de gràficas, tanto de dirigidas como de no dirigidas.



(a) Gráfica no dirigida (b) Gráfica dirigida

Fig 1.1 Gráficas.

De igual manera, para transformar una gráfica no dirigida a otra dirigida se cambian los lados (v,w) de la gráfica no dirigida por las que pueden existir en la gráfica dirigida, como son [v,w] y [w,v].

1.1.2. RUTAS MINIMAS.- ALGORITMO DE DIJKSTRA.

Uno de los problemas relacionados con gráficas y árboles que tiene un buen número y diverso tipo de aplicaciones es determinar la ruta de menor longitud entre dos vértices cualquiera de una gráfica.

Sea G una gráfica dirigida cuyos lados tienen asignados valores que representan la longitud o factor de peso para transitar desde un vértice al otro en el lado. Se denominará la longitud de un lado [v,w] como length (v,w). La longitud de una ruta p. simbolizada como length (p) es la suma de las longitudes de los lados de p. Una ruta mínima desde un vértice **s** a un vértice **t** es una ruta desde **s** a **t** tal que su longitud es mínima.

Algoritmo de Dijkstra:

El algoritmo de Dijkstra es posiblemente el más frecuentemente utilizado para determinar la ruta más corta desde un vértice de una gráfica hasta los restantes vértices.

El algoritmo de Dijkstra etiqueta los vértices de la gráfica. En cada paso del algoritmo algunos vértices quedan etiquetados permanentemente y otros temporalmente, los vértices que quedan etiquetados permanentemente tienen ya su ruta mínima desde un vértice inicial s.

El siguiente es el algoritmo:

0. se asigna una etiqueta permanente de 0 para el vértice inicial s y una etiqueta temporal de infinito (∞) para los restantes vértices de la gráfica.

A partir de este punto, en cada etapa un nuevo vértice alcanza su etiqueta permanente de acuerdo a las siguientes reglas:

1. Cada vértice j que no está aún etiquetado permanentemente alcanza una nueva etiqueta temporal cuyo valor viene dado por:

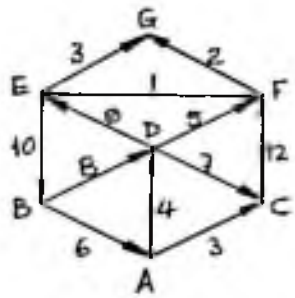
min : [dist j, (dist i + length (i,j))], donde,

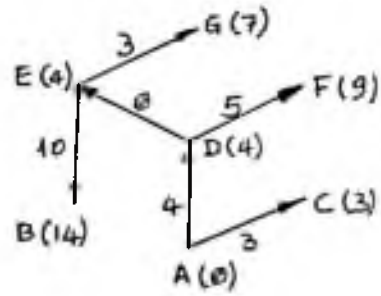
i = último vértice etiquetado permanentemente en la iteración anterior. Los vértices j son los restantes vértices.

2. El valor más pequeño de entre las etiquetas temporales asi encontradas viene a ser la etiqueta permanente para ese vértice correspondiente. En caso de que más de un vértice coincida en ese menor valor se selecciona a cualquiera de esos vértices con etiqueta permanente.

Los pasos 1 y 2 se repiten alternadamente hasta que el vértice de destino t alcance su etiqueta permanente.

Ejemplo 1.1 Para la siguiente gráfica dirigida determinar las longitudes mínimas de ruta desde el vértice A, utilizando el algoritmo de Dijkstra.





A	B	C	D	E	F	G	
0*	∞	∞	∞	∞	∞	∞	paso 0
0*	∞	3	4	∞	∞	∞	paso 1
0*	∞	3*	4	∞	∞	∞	paso 2
0*	∞	3*	4	∞	15	∞	paso 1
0*	∞	3*	4*	∞	15	∞	paso 2
0*	∞	3*	4*	4	9	∞	paso 1
0*	∞	3*	4*	4*	9	∞	paso 2
0*	14	3*	4*	4*	9	7	paso 1
0*	14	3*	4*	4*	9	7*	paso 2
0*	14	3*	4*	4*	9*	7*	paso 2
0*	14*	3*	4*	4*	9*	7*	paso 2

Estas son las longitudes de ruta mínimas desde el vértice A hasta cada uno de los vértices restantes B,C,D,E,F y G.

1.2. NORMAS DE MEDIDA.

Existen diversos criterios para medir la distancia entre dos puntos, es de interes para el presente trabajo el revisar dos de estos criterios : la norma l_1 y la norma l_∞.

1.2.1. NORMA l_1.

En la norma l_1 , la distancia entre dos puntos del plano es igual a la suma de los valores absolutos de la diferencia de abscisas y de la diferencia de ordenadas existentes entre los dos puntos considerados. Es decir la distancia entre dos puntos, P1 y P2 dados por sus coordenadas, medida con norma l_1 se obtiene con la expresión:

$$d = |X2 - X1| + |Y2 - Y1|$$

La norma l_1 , aplicada consecutivamente a una sucesión de puntos intermedios entre un punto inicial y un punto final da lugar a la denominada "ruta de Manhattan". Esta ruta esta constituida entonces por segmentos de recta horizontales y verticales exclusivamente.

1.2.2. NORMA l infinito (l_∞).

La norma l_∞ es aquella en la que se considera que la distancia d entre dos puntos P1 y P2, dados por sus coordenadas es igual al mayor valor de entre los valores absolutos de la diferencia de abscisas y de la diferencia de ordenadas, existentes entre los dos puntos. La expresion de cálculo es:

$$d = \max (|X2 - X1| , |Y2 - Y1|)$$

Se analiza más detenidamente esta definición.

Se definió la norma l_1 , en la cual la distancia d entre dos puntos P1 y P2, dados por sus coordenadas es:

$$d = |X2 - X1| + |Y2 - Y1|$$

De igual manera la norma l_2 para calcular la distancia entre dos puntos es:

$$d = \sqrt{(X2 - X1)^2 + (Y2 - Y1)^2}$$

Si se extiende este concepto existe la norma l_n segun la cual la distancia entre los dos puntos es:

$$d = \sqrt[n]{(X2 - X1)^n + (Y2 - Y1)^n}$$

Si se supone que n es un número muy grande se tiene la norma

l_∞.

Se puede modificar la última expresión como sigue, asumiendo además que

$$|X2 - X1| > |Y2 - Y1|$$

$$d = |X2 - X1| \sqrt[n]{1 + \frac{(Y2 - Y1)}{(X2 - X1)}^n} = |X2 - X1| \sqrt[n]{1 + \frac{|Y2 - Y1|}{|X2 - X1|}}$$

donde $\frac{|Y2 - Y1|}{|X2 - X1|} < 1$; la expresión $\left(\frac{|Y2 - Y1|}{|X2 - X1|}\right)^n \dashrightarrow 0$

Y se obtiene para el valor de la distancia d :

$$d = |X2 - X1|,$$

que corresponde al mayor valor entre $|X2 - X1|$ y $|Y2 - Y1|$

A diferencia de lo que se indicó para el uso de la norma l_1 que considera segmentos de recta horizontales y verticales, la norma l_∞ puede considerar adicionalmente segmentos de recta oblicuos a 45 grados.

La norma l_∞ es un concepto muy utilizado en el diseño de circuitos integrados, al igual que la norma l_1 .

El algoritmo de enrutamiento que se propone en el presente trabajo se ha desarrollado para que pueda obtener la distancia más corta entre dos puntos tanto con la norma l_1 como con la norma l_∞.

CAPITULO 2

LA GRAFICA DE FIGURAS.

En este capítulo se introduce uno de los conceptos importantes de este trabajo y que se lo ha denominado "gráfica de figuras", es decir el conjunto de figuras geométricas y su concepción como una gráfica , en que se divide el plano original como consecuencia de la introducción de los obstáculos en el plano.

2.1 OBSTACULOS Y SU REPRESENTACION.

El espacio bajo consideración es el plano referenciado con un sistema de ejes coordenados X,Y.

Sea T el conjunto de rutas u obstáculos ya establecidos en el plano de referencia, cada elemento de T es un segmento de recta horizontal, vertical u oblicuo a 45 o 135 grados, definido por los siguientes parámetros:

a) un punto inicial definido por sus coordenadas (x_i, y_i) y,

b) un punto final definido igualmente por sus coordenadas (x_f, y_f).

En los obstáculos horizontales el punto inicial es el izquierdo y el final es el derecho, en los obstáculos verticales el punto inicial es el inferior y el final es el superior, en los obstáculos oblicuos el punto inicial es el de ordenada inferior y el final es el de ordenada superior.

Con el fin de poder manejar con mayor facilidad a los obstáculos se asigna a estos un número que identifica el tipo de recta que se esta utilizando en un instante dado. La convención para identificar a una recta por su tipo es :

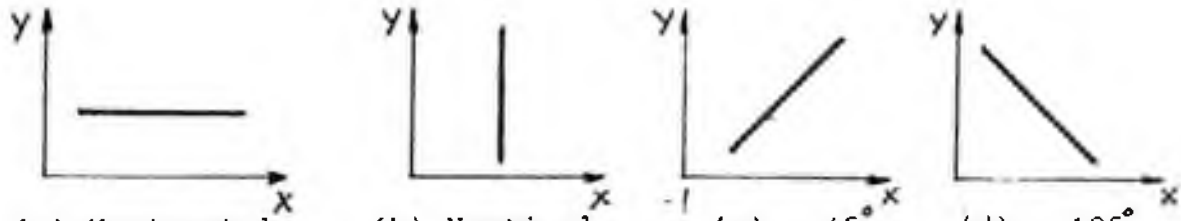
0 si se trata de una recta horizontal,

1 si se trata de una recta vertical,

2 si se trata de una recta oblicua a 45 grados y,

3 si se trata de una recta oblicua a 135 grados.

En la figura 2.1 se muestra un conjunto de rectas clasificadas de acuerdo a su tipo.



(a) Horizontal tipo 0 (b) Vertical tipo 1 (c) a 45° tipo 2 (d) a 135° tipo 3

Fig 2.1 Tipos de rectas

Los obstáculos pueden estar aislados o coincidiendo en uno de sus extremos en cuyo caso se tendría un polígono abierto. Adicionalmente pueden también considerarse como obstáculos a polígonos cerrados los mismos que serán tratados como un conjunto de rectas (los lados del polígono).

2.2 TIPOS DE FIGURAS GEOMETRICAS Y SU REPRESENTACION.

Los tipos de figuras que se considerarán en la gráfica de figuras son cuadriláteros y triángulos.

Los cuadriláteros corresponden a las siguientes figuras :

- Rectángulos
- Trapecios rectángulos
- Trapecios isósceles
- Romboides

Todos los triángulos son rectángulos.

La representación general de estas figuras geométricas será en base a las coordenadas de sus vértices. Se asume el siguiente ordenamiento en sentido horario para los vértices :

a) En los cuadriláteros el primer vértice es el inferior izquierdo, el segundo el superior izquierdo, el tercero el superior derecho y el cuarto el inferior derecho.

b) En los triángulos el primer vértice es el de ordenada menor y si existen dos vértices con esta propiedad se escoge el izquierdo como primero. Los restantes dos vértices se ordenan siguiendo el sentido horario de la figura. La manera de representación de los triángulos es la misma que para los cuadriláteros, lo que ocurre es que en el caso de los triángulos no se utiliza uno de los datos de puntos, sino simplemente tres.

Al igual que lo efectuado con los obstáculos se asigna un número de identificación a cada figura, dependiendo de la figura que se trata, de acuerdo a la siguiente convención :

0 para rectángulos,

1 para trapecios rectángulos izquierdos,

2 para trapecios rectángulos derechos,

3 para trapecios isósceles,

4 para romboides agudos,

5 para romboides obtusos,

6 para triángulos rectágulos normales y,

7 para triángulos rectángulos isósceles.

En la figura 2.2 se indican las diferentes figuras geométricas consideradas y el tipo asignado a ellas.

No siempre las figuras geométricas así definidas se encntrarán en la posición mostrada en la Fig. 2.2, sino que pueden aparecer rotadas a 90, 180 o 270 grados. Por esta razón es también necesario asignar a cada figura un número que indique cual es la posición en la que se encuentra, de conformidad con la siguiente convención :

0 si se encuentra en la posición normal (la de la Fig. 2.2),

1 si se encuentra rotada 90 grados en sentido horario,

2 si se encuentra rotada 180 grados en sentido horario y,

3 si se encuentra rotada 270 grados en sentido horario.

En la Fig. 2.3 se muestran las figuras geométricas en las diferentes posiciones en las que pueden encontrarse, se observa que el rectángulo es la única figura que es independiente de la posición pues siempre tendrá dos lados horizontales y dos lados verticales.

Finalmente, al menos un lado de la figura debe coincidir con todo o parte de un obstáculo, para identificar si un lado coincide con un obstáculo se asigna el número 1 a ese lado, de otra manera si el lado no es obstáculo se asigna un 0 a ese lado.

Resumen.

Cada figura geométrica de las consideradas se representa mediante loa puntos de sus vértices, dados por sus coordenadas y ordenados en sentido horario a partir del vértice inferior izquierdo. Existe un número que identifica el tipo de figura, un número que indica cual es la posición de la figura (normal o rotada) y; respecto a los lados de la figura, si coincide con un obstáculo se asigna un 1 al lado. de lo contrario se asigna un 0.

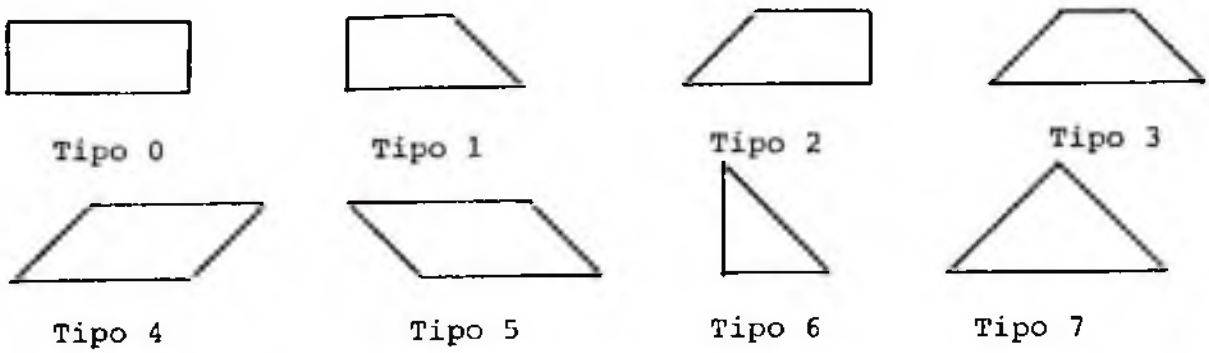
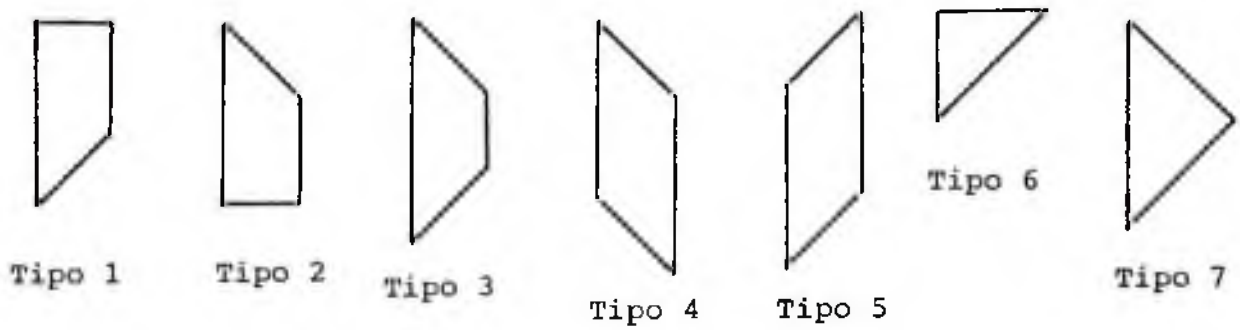
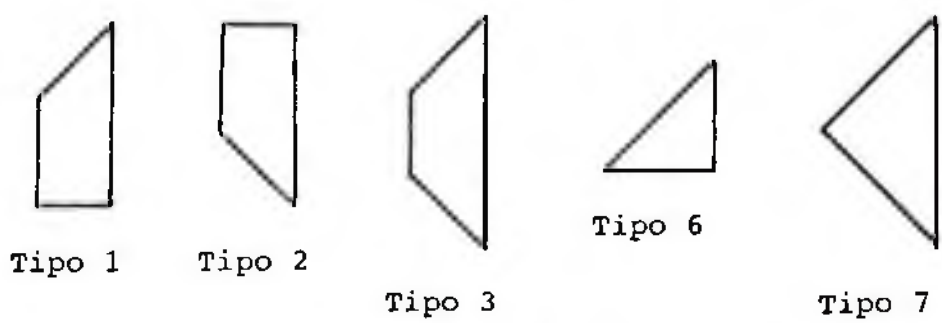


Fig. 2.2 Tipos de figuras geométricas.



(a) Rotada a 90 grados.

(b) Rotada a 180 grados.



(c) Rotada a 270 grados.

Fig. 2.3 Posiciones de las figuras geométricas.

2.3 CONSTRUCCION DE LA GRAFICA DE FIGURAS.

2.3.1 GENERACION DE LAS FIGURAS DE LA GRAFICA.

La gráfica de figuras se construye siguiendo un procedimiento repetitivo y uniforme conforme se van introduciendo uno a uno los obstáculos. El orden en que se introducen los obstáculos es irrelevante aunque un determinado orden puede ser más favorable que otro. El procedimeiento que se sigue con cada obstáculo esta encaminado a dividir en otras figuras geométricas a la figura en la cual se encuentra el punto inicial del obstáculo. Los aspectos importantes con los que se efectúa esta división son los siguientes :

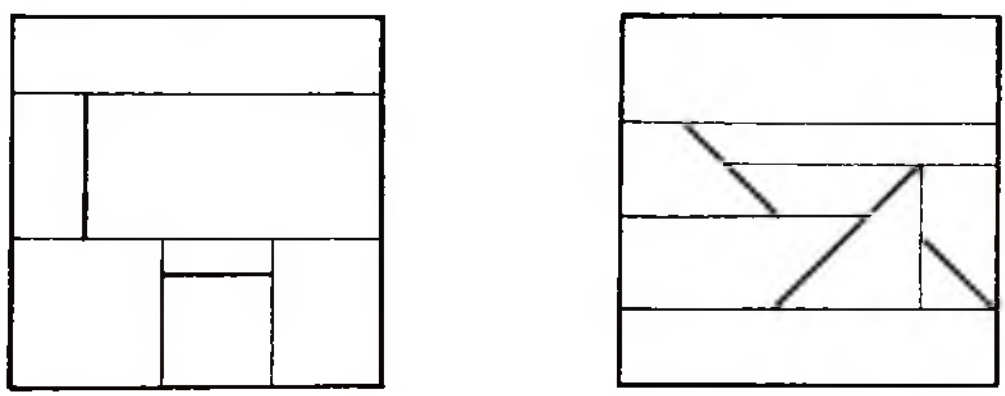
a) se trazan líneas perpendiculares por los extremos de cada uno de los segmentos rectilíneos horizontales y verticales correspondientes a obstáculos de esos tipos.

b) se trazan líneas horizontales o verticales por los extremos de los segmentos rectilíneos correspondientes a los obstáculos oblicuos, sean a 45 o 135 grados. No se trazan lineas perpendiculares por los extremos de los obstáculos oblicuos para asegurar que las particiones de la figura sigan siendo a lo más cuadriláteros.

c) se extienden estas líneas trazadas por los extremos de los obstáculos en los dos sentidos hasta encontrar una de las siguientes limitaciones :

- los límites del espacio de estudio.
- a otro obstáculo.
- otras líneas similares construidas previamente con relación a otros obstáculos.

En la figura 2.4 se ilustra la manera de efectuar la división de una figura.



(a) por un obstáculo vertical u horizontal. (b) por un obstáculo oblicuo

Fig. 2.4 Proceso de división de una figura.

d) se efectúan uniones de dos o mas figuras en una sola siempre que la figura resultante continúe perteneciendo al conjunto de figuras definido en la sección anterior. Claramente, estas uniones pueden efectuarse a través de los lados ficticios de las figuras generados mediante lo descrito en a) y b).

e) se determinan las figuras adyacentes con las que una figura tiene comunicación, a través de aquellos lados que no son obstáculos, a estas figuras adyacentes se las denominará figuras vecinas de una figura dada. Se liga a la figura considerada el conjunto de sus figuras vecinas. Las partes de la figura original, así conformadas finalmente, substituyen a la figura que se ha dividido.

En consecuencia, un elemento o nodo de la gráfica de figuras estará constituido por la figura en sí y por una lista de figuras vecinas. Cada figura tendrá como representación la descrita en la sección anterior de este capítulo.

f) se actualizan las listas de vecinos de las figuras en las cuales aparezca la figura que se ha dividido. En esas listas una o varias partes de la figura dividida van a substituir a esa figura original.

2.3.2 OBSERVACIONES RELATIVAS A LA DIVISION DE FIGURAS.

En el proceso descrito en la sección anterior, relativo a la división de figuras pueden darse las siguientes situaciones:

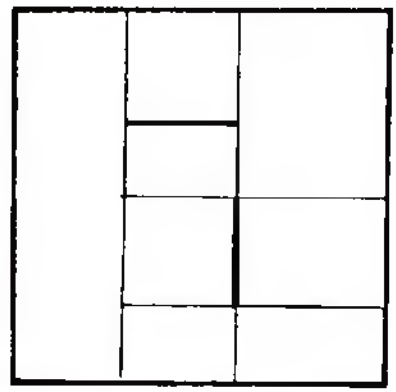
a) el número de partes en las que un obstáculo divide a la figura en la que se encuentra el punto inicial depende sobretodo de si el obstáculo esta contenido o no íntegramente en esa figura, influye también el tipo de figura que se va a dividir. El número de partes en las que una figura se divide puede ser dos, tres, cuatro o cinco, sin embargo, lo más frecuente son las divisiones en dos, tres o cuatro partes; la división en cinco partes es una situación muy particular. En la Fig. 2.7 se muestran la variedad de casos de división que pueden presentarse dependiendo del tipo de obstáculo y del tipo de figura.

b) se efectúan los pasos descritos para la división solamente con figuras en la posición normal; si la figura se encuentra en una posición de rotación se modifica previamente, tanto la figura como el obstáculo con un sencillo proceso de rotación de ejes, a fin de pasar de la posición de rotación a la posición normal. Una vez completada la división de la figura se aplica otro proceso de rotación para volver a las partes a su posición original de rotación. De esta manera se reducen los casos de división.

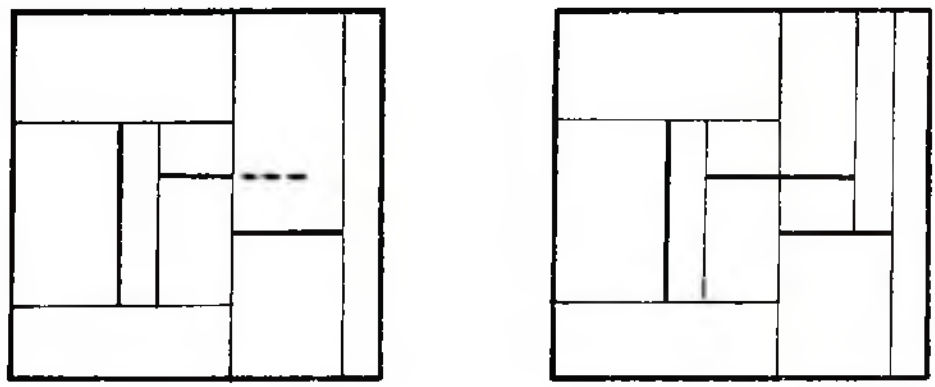
c) si un obstáculo coincide con un lado de la figura a la que va a dividir, es decir es común a dos figuras deben dividirse estas dos figuras independientemente. La Fig. 2.5 a) muestra este caso.

d) Si el punto final del obstáculo se encuentra fuera de la figura que actualmente se esta dividiendo, debe aplicarse el

proceso de división a la o las figuras adyacentes a las que afecta el mismo obstáculo hasta que se considere una figura para la cual el punto final del obstáculo sea interior. La Fig. 2.5 b) muestra este caso.



(a) El obstaculo coincide con un lado de figuras adyacentes.

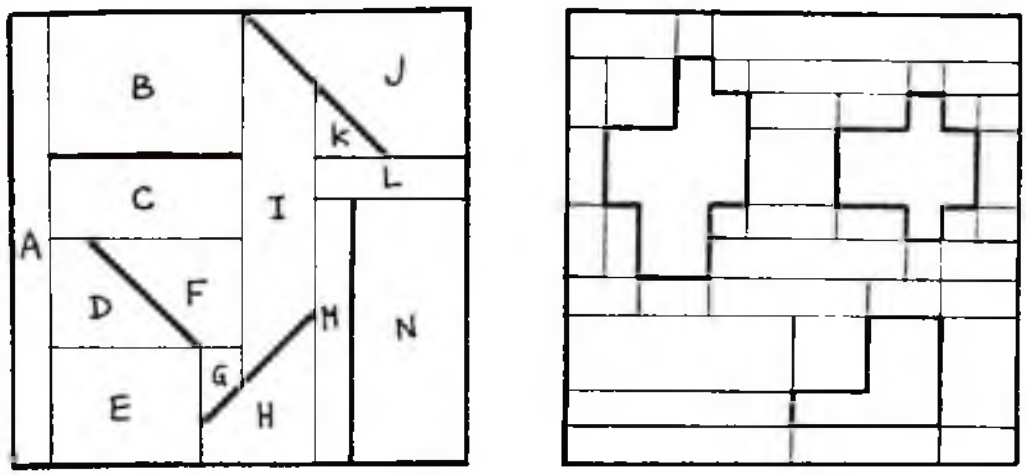


(b) El obstáculo empieza en una figura y termina en otra.

Fig. 2.5 Situaciones de división de más de una figura por el mismo obstáculo.

e) si además de los obstáculos lineales existen adicionalmente obstáculos constituidos por polígonos cerrados el procedimiento para generar las figuras geométricas es enteramente similar; simplemente se aplica el proceso de división a cada lado del polígono, de manera independiente.

En la figura 2.6 se muestra un plano y su división en regiones geométricas cuando se tienen obstáculos lineales y también obstáculos constituidos por polígonos.



(a) (b)

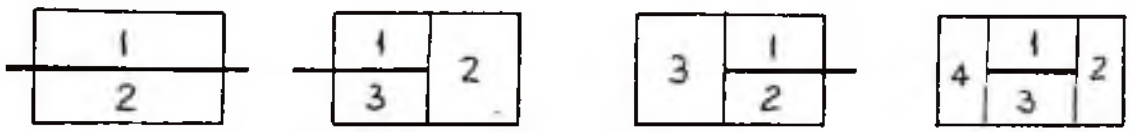
Fig. 2.6 Conformación de las regiones geométricas.
(a) solo con obstáculos lineales,
(b) añadiendo también poligonos.

2.3.3 NUMERO DE FIGURAS GEOMETRICAS QUE SE GENERAN.

Respecto al número total de figuras que se generan una vez introducidos todos los obstáculos, salvo el caso particular de división de una figura en cinco partes la situación general será la de división de una figura en a los más cuatro partes. Bajo este criterio se puede probar que el número de figuras que se generan con este procedimiento, a partir de m obstáculos es a lo más de $3m + 1$, pues, si existe un solo obstáculo se generan inicialmente a lo más 4 figuras, y, por cada obstáculo adicional se generan a lo más 3 figuras adicionales. En el punto 2.3.4 se establecen las situaciones en las que se pueden presentar casos de división de una figura en cinco partes.

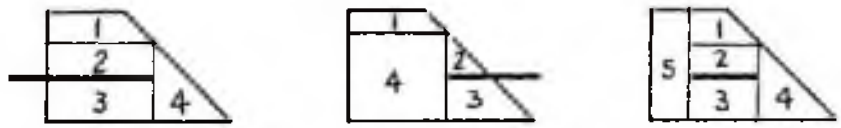
2.3.4 CASOS DE DIVISION DE FIGURAS.

En la Fig. 2.7 se muestran los diferentes casos de división que pueden presentarse en la división de figuras. Como puede apreciarse existen divisiones aplicables a todas las figuras (divisiones estandar) y también casos especiales, por ejemplo la división de una figura en cinco partes se presenta solamente cuando se divide un trapecio o un triángulo isósceles por un obstáculo horizontal cualquiuera en el caso del triángulo o, que sobrepase los limites de la base menor en el caso del trapecio.

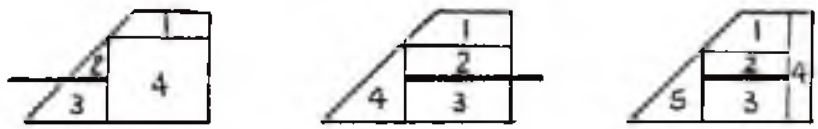


2 partes 3 partes 3 partes 4 partes

Divisiones estandar (aplicables a toda figura)

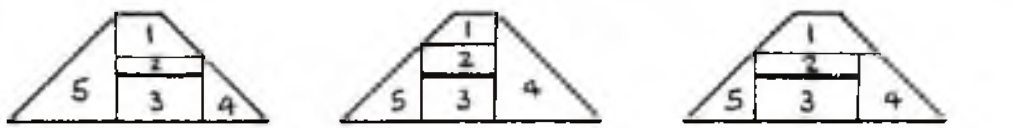


4 partes 4 partes 5 partes

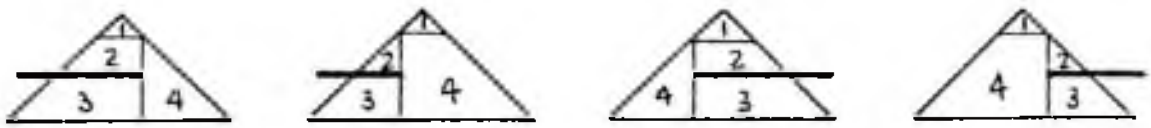


4 partes 4 partes 5 partes

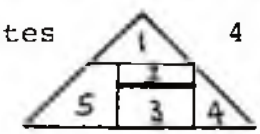
4 partes 4 partes 4 partes 4 partes



5 partes 5 partes 5 partes



4 partes 4 partes 4 partes 4 partes

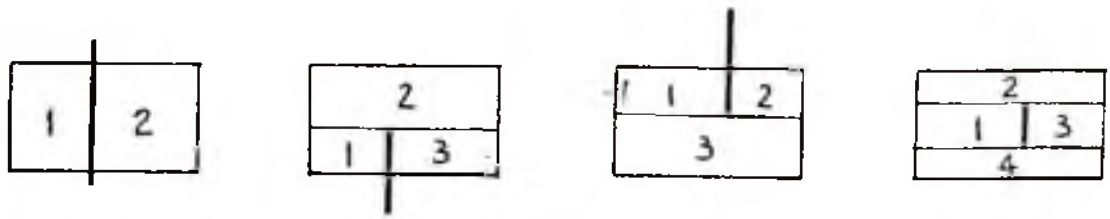


5 partes

Casos de divisiones especiales

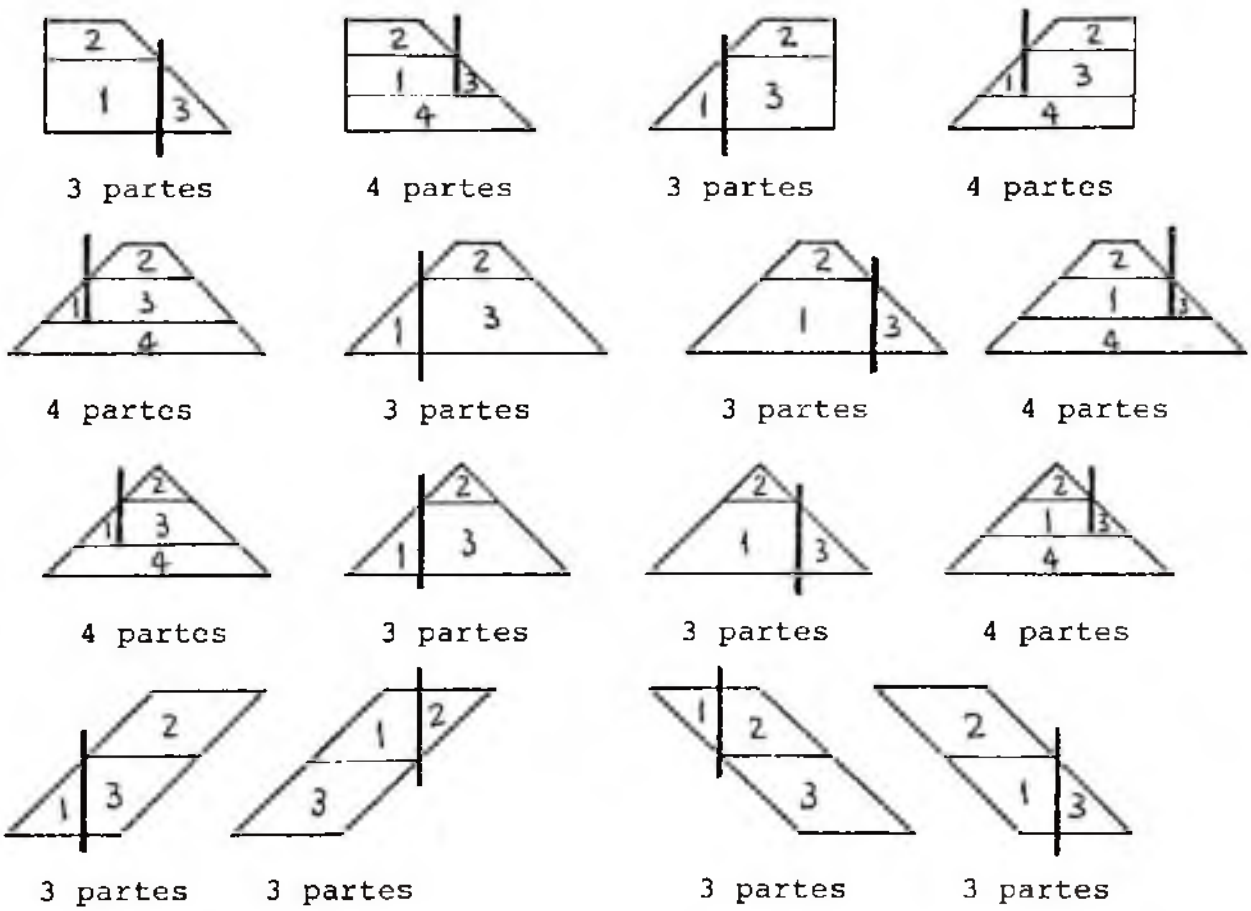
Fig. 2.7 Divisio'n de figuras.

(a) Efectos de obstáculos horizontales.



2 partes 3 partes 3 partes 4 partes

Divisiones estandar (aplicable a toda figura)



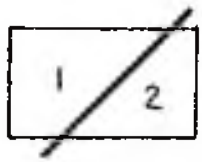
3 partes 4 partes 3 partes 4 partes

4 partes 3 partes 3 partes 4 partes

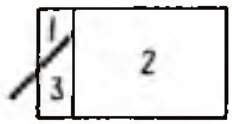
4 partes 3 partes 3 partes 4 partes

3 partes 3 partes 3 partes 3 partes

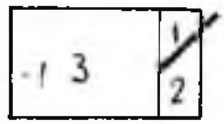
Casos de divisiones especiales.
Fig. 2.7 División de figuras.
(b) Efectos de obstáculos verticales.



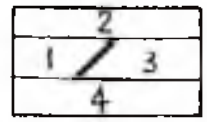
2 partes



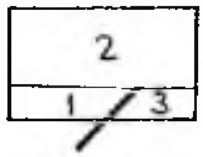
3 partes



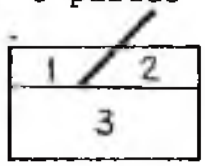
3 partes



4 partes

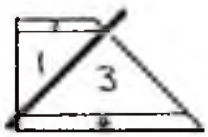


3 partes



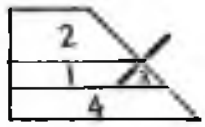
3 partes

Divisiones estandar (aplicables a toda figura)

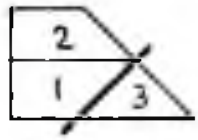


4 partes

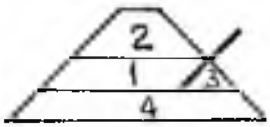
4 partes



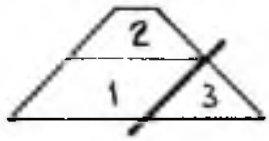
4 partes



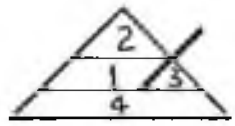
3 partes



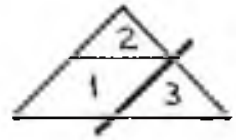
4 partes



3 partes



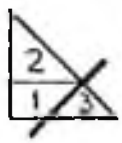
4 partes



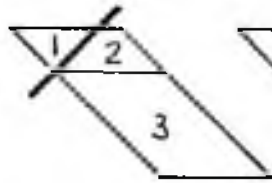
3 partes

3 partes

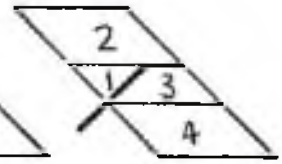
4 partes



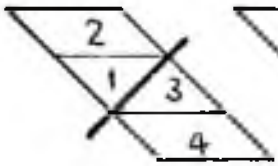
3 partes



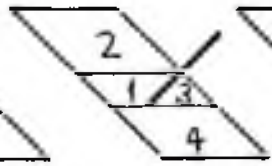
3 partes



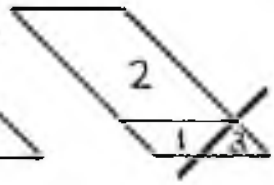
4 partes



4 partes



4 partes

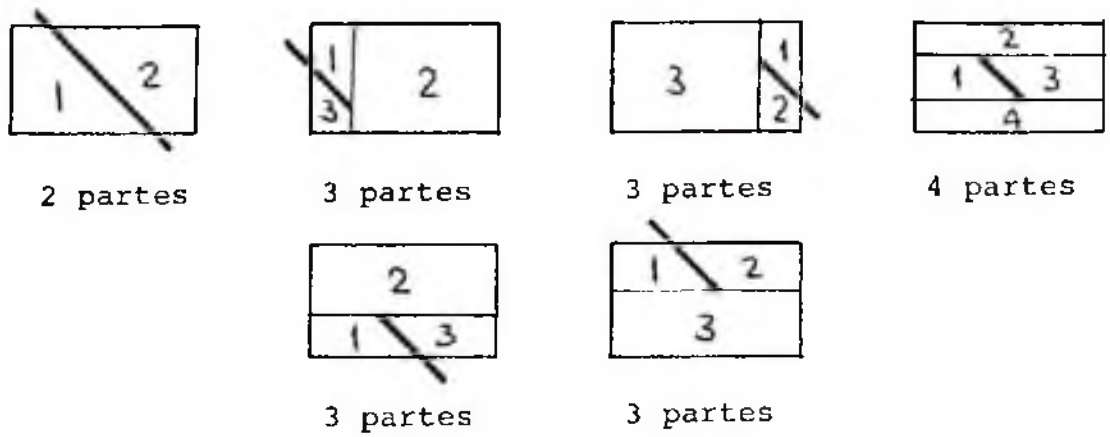


3 partes

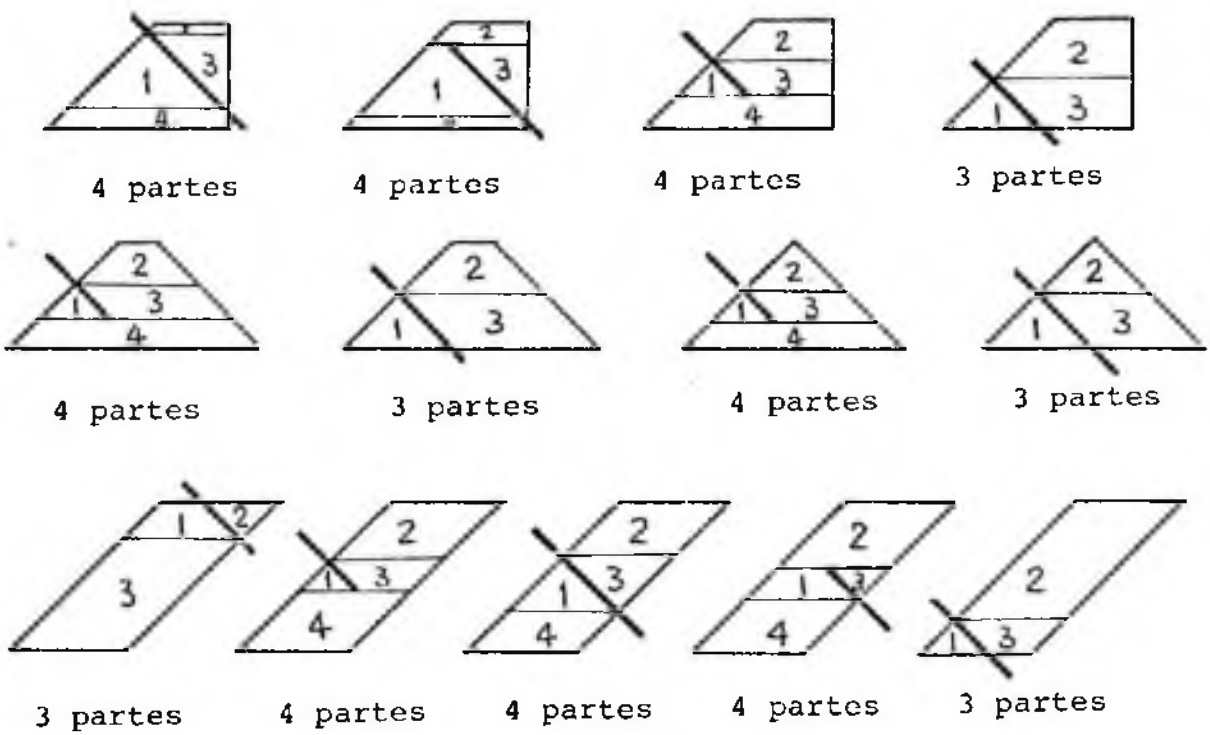
Casos de divisiones especiales.

Fig. 2.7 División de figuras.

(c) Efectos de obstáculos oblicuos a 45 grados.



Divisiones estandar (aplicable a toda figura)



Casos de divisiones especiales.
Fig. 2.7 División de figuras.
(d) Efectos de obstáculos oblícuos a 135 grados.

2.4. EL MODELO DE GRAFICA A PARTIR DE LAS FIGURAS GEOMETRICAS.

Una vez que se ha formado el conjunto de figuras geométricas mediante el procedimiento descrito anteriormente puede efectuarse la correspondencia del plano asi particionado en figuras con una gráfica no dirigida en la siguiente manera:

- cada figura geométrica corresponde a un vértice de la gráfica.

- un vértice tiene relación con otro si en las figuras correspondientes una figura es adyacente con otra a través de un lado ficticio, es decir los lados de figuras que representan obstáculos no expresan relación o comunicación con una figura adyacente. En base a esta consideración se construyen las aristas o lados de la gráfica.

En este trabajo la gráfica de figuras esta constituida como una lista ligada donde cada nodo es una figura geométrica del plano, la cual, como se recordará tiene asociada una lista de figuras vecinas.

En la figura 2.8 se indica la gráfica correspondiente a la división en figuras de la figura 2.6.

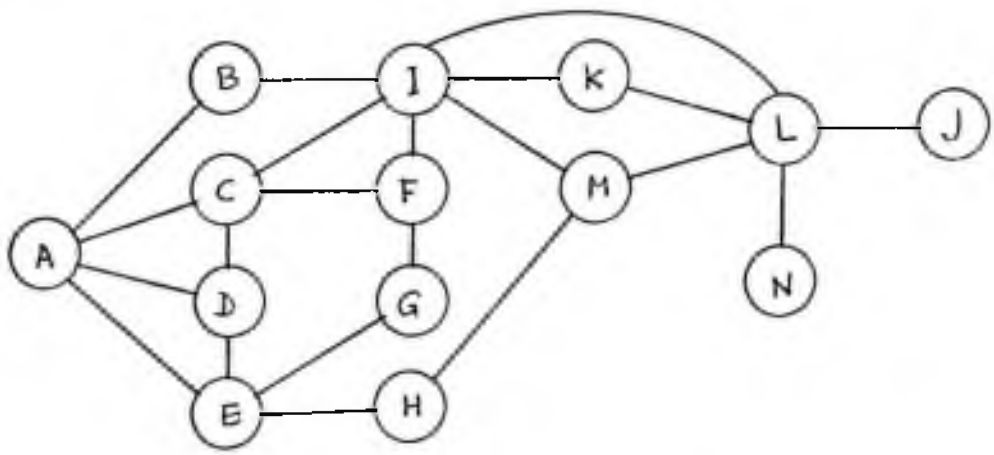


Fig. 2.8 Gráfica representativa de las figuras geométricas.

2.5 ALGORITMOS UTILIZADOS PARA LA CONSTRUCCION DE LA GRAFICA DE FIGURAS.

Tal como se ha descrito en la sección 2.3 el procedimiento general de formación de la gráfica de figuras consta de tres etapas bien definidas:

a) la división de una figura en sus partes,

b) la obtención de la lista de figuras vecinas para cada parte y,

c) la actualización de la gráfica en general como consecuencia de

la inclusión de las partes y la supresión de la figura que se dividió.

A continuación se presenta el algoritmo general de generación de la gráfica de figuras, así como los relativos a las etapas descritas.

2.5.1 ALGORITMO GENERAL PARA LA CONSTRUCCION DE LA GRAFICA.

Mientras se ingresan obstáculos se efectúa el siguiente procedimiento:

1. Se ingresan los datos de los puntos extremos del obstáculo.

2. Se obtiene el apuntador de la figura en la que empieza el obstáculo.
   Si el obstáculo es común a dos figuras se obtiene el apuntador de la segunda figura en la cual empieza también el obstáculo.
   Para cada figura determinada por su apuntador se aplican los siguientes pasos, del 3 al 8:

3. Se efectúa la división de la figura obteniéndose las partes correspondientes a la figura original.

4. Se obtiene la lista de figuras vecinas para cada una de las partes

5. Se insertan las partes en la estructura general de la gráfica en el sitio determinado por el apuntador a la figura original.

6. Se actualizan las listas de figuras vecinas para aquellas figuras de la gráfica que tienen como vecina a la figura original.

7. Se elimina de la gráfica la figura que se dividió.

8. Si el obstáculo no continúa en otra figura se termina con el procedimiento.
   Si el obstáculo continúa en otra figura se busca en la lista de figuras vecinas cual es la siguiente figura a dividirse, se obtiene su apuntador y se repiten los pasos del 3 al 8 inclusive.

2.5.2 ALGORITMO QUE DIVIDE UNA FIGURA EN SUS PARTES.

1. Dependiendo del tipo de obstáculo introducido se selecciona el procedimiento adecuado de división de figuras.

2. Si la figura esta en su posición normal se pasa al siguiente paso.
   Si la figura esta en una posición de rotación se transforman por rotación tanto la figura como el obstáculo a la posición normal.

3. Se aplica el procedimiento que obtiene las partes de la

figura.

4. Si la figura original estuvo en posición normal se termina con el algoritmo.
   Si la figura original fue transformada por rotación a la posición normal se aplica a las partes la transformación inversa de rotación para que las partes correspndan a la posición original de la figura que se ha dividido.

2.5.3 ALGORITMO QUE OBTIENE LA LISTA DE VECINOS DE UNA FIGURA.

La lista de figuras vecinas de una figura correspondiente a una de las partes de una figura que se ha dividido se obtiene en dos etapas:

a) En una primera etapa se obtienen las figuras vecinas de entre las restantes partes de la figura original, a esta lista se denomina lista interna de vecinos.

b) En una segunda etapa se añaden a la lista obtenida en a) las figuras que son vecinas, de entre la lista de vecinos de la figura original.

La Fig 2.9 muestra la denominación de figuras vecinas.

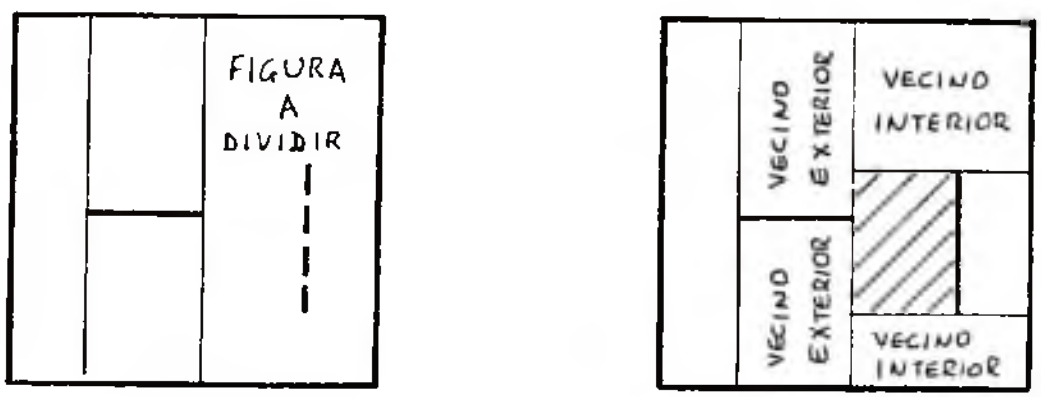


Fig. 2.9 Definición de figuras vecinas.

Los algoritmos correspondientes a las dos etapas son:

2.5.3.1 ALGORITMO PARA OBTENER LA LISTA INTERNA DE VECINOS.

Para cada una de las partes se siguen los siguientes pasos:

1. Se toman de uno en uno los lados de la figura, si coinciden con un obstáculo se pasa al siguiente lado, si no es obstáculo se pasa al siguiente paso.

2. Se compara el lado seleccionado en el paso 1. con los lados que no son obstáculos de las restantes partes.

Si dos lados son comunes total o parcialmente se añade a la lista de vecinos la figura correspondiente.

2.5.3.2 ALGORITMO QUE COMPLETA LA LISTA DE VECINOS DE UNA FIGURA.

Para cada una de las partes se siguen los siguientes pasos:

1. Se toman de una en una las figuras vecinas de la figura original.
   Se toman de uno en uno los lados de estas figuras. Si corresponden a obstáculos se pasa al siguiente lado; si no son obstáculos se pasa al siguiente paso.

2. Se compara el lado seleccionado en el paso 1. con los lados que no son obstáculos de la parte de la cual se esta obteniendo su lista de vecinos.
   Si dos lados son comunes total o parcialmente se añade a la lista de vecinos la figura correspondiente.

2.5.4 ALGORITMO QUE ACTUALIZA LAS LISTAS DE VECINOS DE LAS FIGURAS QUE SON VECINAS A LA FIGURA QUE SE DIVIDIO.

Se accede a la lista de figuras vecinas de la figura que se ha dividido (a la cual se identifica por **f**) y para cada figura de esta lista (a la cual se identifica por **f1**) se sigue el siguiente proceso:

1. Se busca en la estructura general de la gráfica a la figura considerada (**f1**).

2. Se accede a la lista de figuras vecinas de la figura **f1**.

3. Si en esta lista de vecinos se encuentra la figura original que se dividió (**f**) se efectúan los siguientes pasos, de otro modo se termina con el algoritmo para la figura **f1**.

4. Se toman una a una las particiones de **f** y se recorre su correspondiente lista de figuras vecinas. Si en esta lista aparece la figura **f1** entonces se añade, de manera recíproca, a la lista de vecinos de **f1** la figura correspondiente a la partición.

5. Se elimina de la lista de figuras vecinas de **f1** a la figura original **f** y se termina el algoritmo para la figura **f1**.

2.6 COMPLEJIDAD DE LOS ALGORITMOS.

Si **m** es el número de obstáculos presentes en el plano, el número de figuras geométricas obtenidas es de **O (m)**, pues corresponde a lo más **3m + 1** en el caso usual o en el más desfavorable **4m + 1**.

La gráfica de figuras puede almacenarse utilizando diversas

estructuras de datos. Posiblemente las estructuras más convenientes que se pueden utilizar son las de árbol y entre estos los **árboles R** [6] o los **árboles cuaternarios (quad trees)** [7].

Igualmente se ha indicado que el orden en el que se introduzcan los obstáculos es irrelevante, sin embargo un orden u otro puede causar que se tenga que recorrer un mayor o menor número de figuras conforme se ejecuta el algoritmo general de construcción de la gráfica.

Sea n el número de figuras geométricas de la gráfica de figuras en un instante dado de su generación, al introducir un nuevo obstáculo se tienen los siguientes órdenes de complejidad en los diferentes pasos que se siguen para la obtención de un nuevo nodo de la gráfica:

En la fase de búsqueda de la figura en la cual esta el punto inicial el algoritmo es del orden de O (log(n)).

En la etapa de la obtención de la lista de vecinos puede presentarse la situación más desfavorable que un mismo obstáculo tenga que recorrer hasta n/2 nodos, con lo cual el orden del algoritmo sería de O (n).

En consecuencia, el orden del algoritmo de generación de la gráfica, para n obstáculos es de:

$$O\ (n^2).$$

El orden de O (n^2) es el precio que se paga al introducir los obstáculos de manera totalmente aleatoria y que ese orden aleatorio provoque la complejidad indicada. Sin embargo, en la mayoria de casos reales se tiene la convicción que el orden del algoritmo es en esperanza de O (n).

Cabe señalar finalmente que existen algoritmos con orden O (nlog(n)) los mismos que generan la gráfica en un solo paso y no de manera incremental como se describe en este trabajo.

CAPITULO 3

METODO DE PROPAGACION DE RANURAS PARA DETERMINAR LA RUTA MINIMA.

Se presenta en este capitulo el mètodo para determinar la ruta de longitud mínima entre dos puntos en presencia de obstàculos basado en el concepto de propagaciòn de ranuras, siendo este mètodo el otro concepto importante que se introduce con este trabajo.

3.1 CONCEPTOS PREVIOS.

En el mètodo de enrutamiento que se propone en el presente trabajo frecuentemente va a ser necesario obtener la distancia mínima con relaciòn a un punto que se denominarà origen, a lo largo de todo un lado de una figura.

Este problema se va a incluir dentro de uno màs general como es el de obtener las distancias entre el punto origen y todos los puntos de una recta. Este planteamiento se va a estudiar en dos instancias del mismo:

a) cuando entre el origen y la recta no existen obstàculos y

b) cuando entre el origen y la recta existen obstàculos.

El problema se va a resolver tanto para la norma l_1 como para la norma l_∞.

Con fines explicativos se analizan en los puntos 3.1.1 y 3.1.2 la obtenciòn de funciones que permitan calcular la distancia desde el origen hasta cualquier punto de la recta en las dos instancias antes indicadas.

3.1.1 FUNCIONES DE DISTANCIA DESDE UN PUNTO ORIGEN HASTA UNA RECTA CUANDO NO EXISTEN OBSTACULOS DE POR MEDIO.

Sea un punto origen P dado por sus coordenadas (X_p, Y_p) y una recta definida por sus extremos A y B, de coordenadas (X_a, Y_a) y (X_b, Y_b), respectivamente. Sin pèrdida de generalidad se supone a la recta horizontal, es decir $Y_a = Y_b = Y_r$ e igualmente se supone a P situado por debajo de la recta, es decir $Y_p <= Y_r$, tal como se ilustra en la Fig. 3.1.

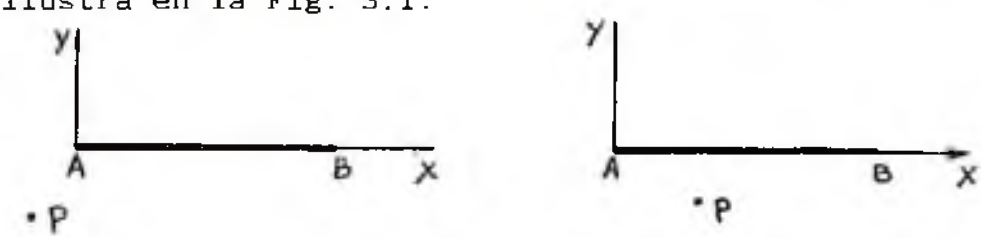


Fig. 3.1 Una recta y su punto origen P

3.1.1.1 FUNCIONES DE DISTANCIA PARA LA NORMA l_1.

Se trata de determinar la función de distancia de los puntos de la recta AB en relación con el punto origen P.

La función de distancia variará conforme se recorra la recta a lo largo de su eje desde el un extremo hasta el otro y dependerá en su expresión algebraica de la posición de P respecto a los extremos de la recta.

La función de distancia, en el intervalo $[X_a, X_b]$, corresponderá a uno de los siguientes tipos:

- Función tipo **f1**, cuando en todo el intervalo es una recta de pendiente negativa.

- Función tipo **f2**, cuando en todo el intervalo es una recta de pendiente positiva.

- Función tipo **f3** cuando en todo el intervalo es una recta paralela al eje x.

- Función tipo **f4** cuando en el subintervalo $[X_a, X_1]$ es del tipo **f1** y en el subintervalo $[X_1, X_b]$ es del tipo f3.

- Función tipo **f5** cuando en el subintervalo $[X_a, X_1]$ es del tipo **f1** y en subintervalo $[X_1, X_b]$ es del tipo f2.

- Función tipo **f6** cuando en el subintervalo $[X_a, X_1]$ es del tipo f3 y en el subintervalo $[X_1, X_b]$ es del tipo f2.

- Función tipo **f7** cuando en el subintervalo $[X_a, X_1]$ es del tipo **f1**, en el subintervalo $[X_1, X_2]$ es del tipo f3 y en el subintervalo $[X_2, X_b]$ es del tipo f2.

En los puntos 3.1.1.1.a y 3.1.1.1.b se deducen las expresiones de la distancia desde un punto origen hasta un punto cualquiera de una recta cuando la recta es horizontal o vertical o cuando es oblicua, respectivamente.

La Fig. 3.2 ilustra estos tipos de funciones.

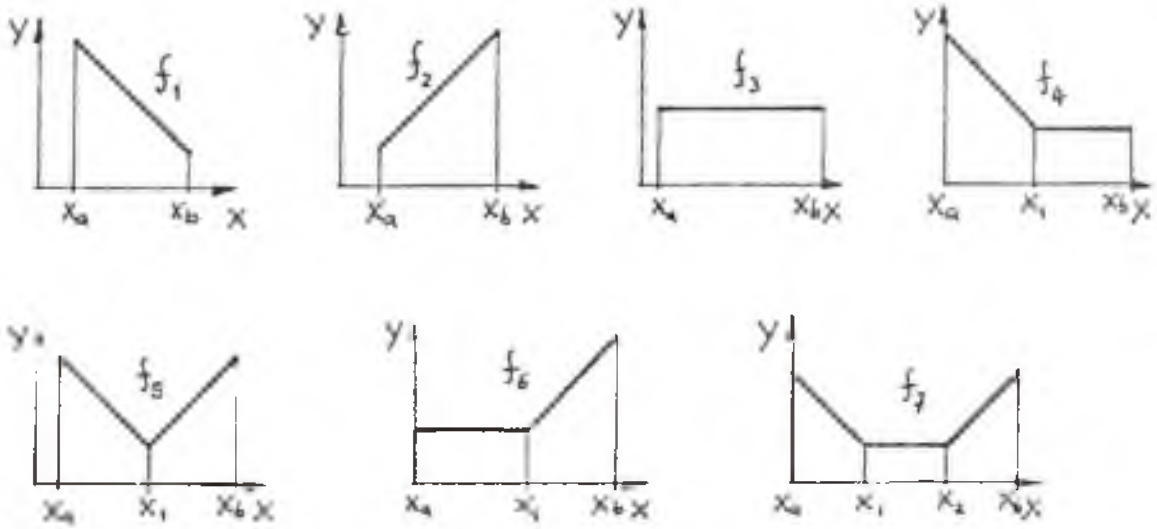


Fig. 3.2 Tipos de funciones de distancia.

3.1.1.1.a LA RECTA ES HORIZONTAL O VERTICAL.

Los tipos de funciones que se tienen aquí son los f1, f2 y f5, exclusivamente. Para las funciones tipo f1 y f2 los valores de las pendientes de las rectas son -1 y +1, respectivamente.

Se consideran los siguientes casos, tomando el caso de una recta horizontal :

1) La ordenada de P es menor o igual que la de la recta, $Y_P <= Y_T$ con las siguientes variaciones de la abscisa del origen:

a) Es menor que la abscisa del extremo izquierdo de la recta:

$$\text{función} = |X_a - X_P| + |Y_P - Y_T| + x$$

b) Esta comprendida entre las dos abscisas de los extremos de la recta:

$$\text{función} = \begin{cases} |X_a - X_P| + |Y_P - Y_T| - x & \text{para } [X_a, X_P] \\ |Y_P - Y_T| - |X_P - X_a| + x & \text{para } [X_P, X_b] \end{cases}$$

c) Es mayor que la abscisa del extremo derecho de la recta:

$$\text{función} = |X_a - X_P| + |Y_P - Y_T| - x$$

Estas funciones se muestran en la Fig. 3.3.

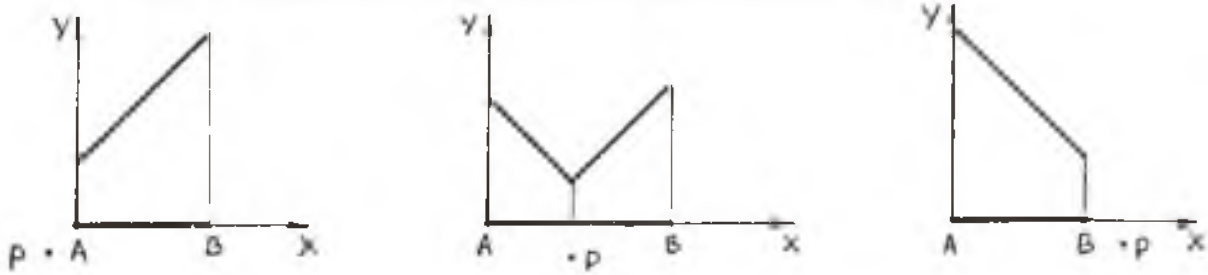


Fig. 3.3 Funciones de distancia desde un origen hasta una recta horizontal.

Expresiones similares y simétricas de funciones pueden obtenerse en los siguientes casos :

2) La ordenada de P es mayor que la de la recta, $Y_p > Y_r$, con las siguientes variaciones de la abscisa del origen:

a) Es menor que la abscisa del extremo izquierdo de la recta.
b) Esta comprendida entre las dos abscisas de los extremos de la recta.
c) Es mayor que la abscisa del extremo derecho de la recta.

En los siguientes dos casos la recta se considera ahora vertical, es decir $X_a = X_b = X_r$, A es el extremo superior y B el inferior de la recta.

3) La abscisa de P es menor o igual que la de la recta, $X_p <= X_r$, con las siguientes variaciones de la ordenada del origen:

a) Es mayor que la del extremo superior de la recta.
b) Esta comprendida entre las dos ordenadas de los extremos de la recta.
c) Es menor que la del extremo inferior de la recta.

4) La abscisa de P es mayor que la de la recta, $X_p > X_r$, con las siguientes variaciones de la ordenada del origen:

a) Es mayor que la del extremo superior de la recta.
b) Esta comprendida entre las dos ordenadas de los puntos de la recta.
c) Es menor que la del extremo inferior de la recta.

3.1.1.1.b LA RECTA ES OBLICUA A 45 O 135 GRADOS.

Los tipos de funciones que se obtienen aquí corresponden a todos los descritos en el punto 3.1.1.1 salvo el tipo f5. Para las funciones tipo f1 y f2 los valores de las pendientes de las rectas son -2 y +2, respectivamente.

La descripción de las diferentes funciones de distancia se efectuará en base a una recta a 45 grados y suponiendo al punto origen ubicado por debajo de la recta.

Se analizan los siguientes casos, en los cuales interesa la variación de las coordenadas del punto origen, tanto en dirección x como en dirección y:

1) La ordenada del origen es menor que la del punto inferior izquierdo de la recta. Para esta situación la abscisa del origen puede variar así:

a) Es menor que la abscisa del extremo inferior izquierdo ($X_p < X_a$) Para el intervalo $[X_a, X_b]$

$$\text{función} = 2x + |Y_a - Y_p| + |X_a - X_p|$$

b) Esta comprendida entre las dos abscisas de los extremos de la recta ($X_a <= X_p <= X_b$).
Para el intervalo $[X_a, X_p]$

$$función = |Y_a - Y_p| + |X_a - X_p|$$

Para el intervalo $[X_p, X_b]$

$$función = 2|x - X_p| + |Y_a - Y_p| + |X_a - X_p|$$

c) Es mayor que la abscisa del extremo superior derecho ($X_p > X_b$).
Para el intervalo $[X_a, X_b]$

$$función = |Y_a - Y_p| + |X_a - X_p|$$

2) La ordenada del origen esta comprendida entre las ordenadas de los puntos extremos de la recta. Para esta situación la abscisa del origen varia asi:

a) Esta comprendida entre las abscisas de los puntos extremos de la recta ($X_a <= X_p <= X_b$).
Para el intervalo $[X_a, X_a + |Y_p - Y_a|]$

$$función = |Y_a - Y_p| + |X_a - X_p| - 2x$$

Para el intervalo $[X_a+|Y_p - Y_a|, X_a+|X_p - X_a|+|Y_p - Y_a|]$

$$función = |Y_p - Y_a| + |X_p - X_a|$$

Para el intervalo $[X_a+|X_p - X_a|+|Y_p - Y_a|, X_b]$

$$función = |Y_p - Y_a| + |X_p - X_a| + 2|x - X_p|$$

b) Es mayor que la abscisa superior derecha ($X_p > X_b$)
Para el intervalo $[X_a, X_a + |Y_p - Y_a|]$

$$función = |Y_p - Y_a| + |X_p - X_a| - 2x$$

Para el intervalo $[X_a + |Y_p - Y_a|, X_b]$

$$función = |Y_b - Y_p| + |X_b - X_p|$$

3) La ordenada del origen es mayor que la del extremo superior derecho de la recta. Para esta situación la abscisa del origen varia asi:
Para el intervalo $[X_a, X_b]$

$$función = |Y_p - Y_a| + |X_p - X_a| - 2x$$

En la Fig. 3.4 se muestran las funciones antes indicadas.

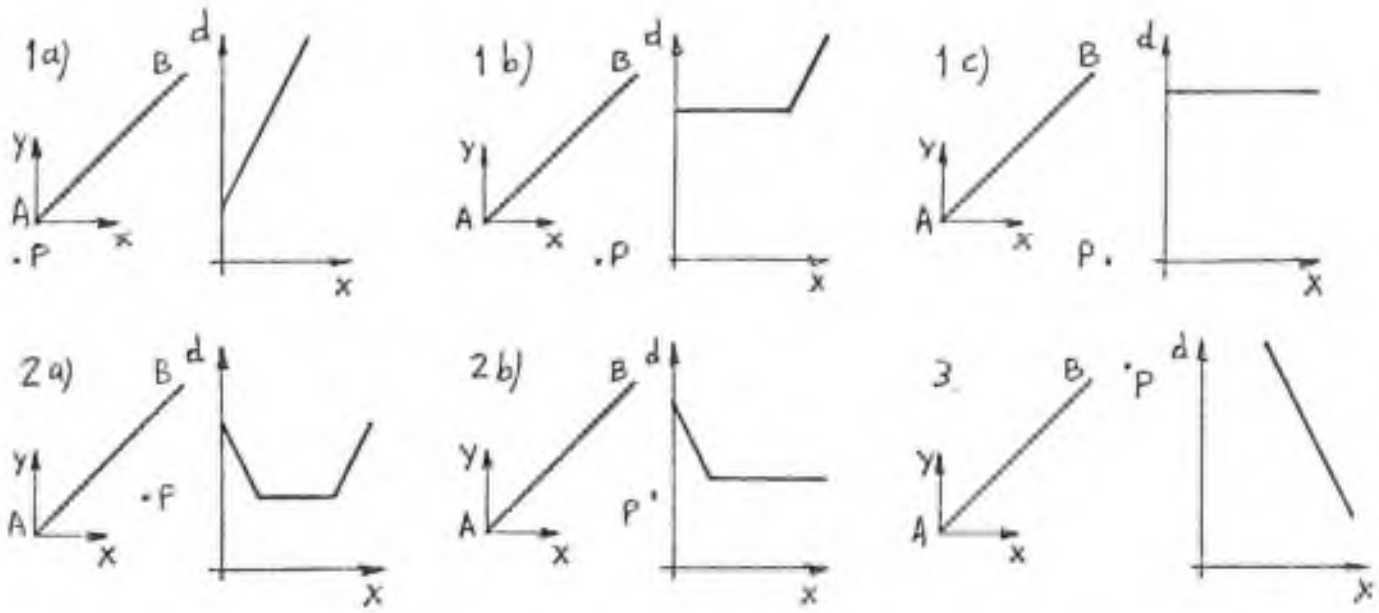


Fig. 3.4 Funciones de distancia desde un origen hasta una recta oblicua.

Con un estudio similar se obtienen las expresiones de las funciones de distancia para cuando el punto origen se ubique por sobre la recta.

Se efectúa también un análisis similar para obtener las funciones de distancia desde un punto origen hasta una recta oblicua a 135 grados.

3.1.1.2 FUNCIONES DE DISTANCIA PARA LA NORMA l_∞.

Las funciones de distancia para los puntos de una recta desde un origen bajo la norma l_∞ se obtienen con un estudio semejante al efectuado para la norma l_1 . Los resultados de ese análisis se indican en el Apéndice A de este trabajo.

3.1.2 FUNCIONES DE DISTANCIA DESDE UN PUNTO ORIGEN HASTA UNA RECTA CUANDO EXISTEN OBSTACULOS DE POR MEDIO.

Las funciones de distancia desde un punto origen hasta una recta, cuando entre el origen y la recta existen obstáculos se expresan en términos de las funciones de distancia obtenidas en la sección 3.1.1 con un proceso adicional de traslado del punto origen hacia los extremos del obstáculo. En otras palabras el punto origen se substituye por dos o más orígenes, cada uno ubicado en los extremos del obstáculo, desde cada uno de estos orígenes se pueden plantear las funciones de distancia ya vistas para cuando no existen obstáculos entre el origen y la recta. Para obtener el valor de la distancia de un punto de la recta respecto al origen inicial se añadirán los valores de la distancia desde el origen inicial hasta en nuevo origen y el valor de la distancia desde el nuevo origen hasta el punto considerado en la recta.

Al haber transformado la obtención de la distancia respecto a un punto origen al cálculo referente a varios orígenes se tendrá una función diferente con su correspondiente dominio para cada origen.

La función de distancia, dividida así en varias, esta constituida de todas maneras por las funciones tipo f1 a f7 vistas anteriormente y es continua en su conjunto.

En la Fig. 3.5 se ilustra el proceso aquí descrito referente a la consideración de varios orígenes, así como las diferentes funciones de distancia respecto a cada origen.

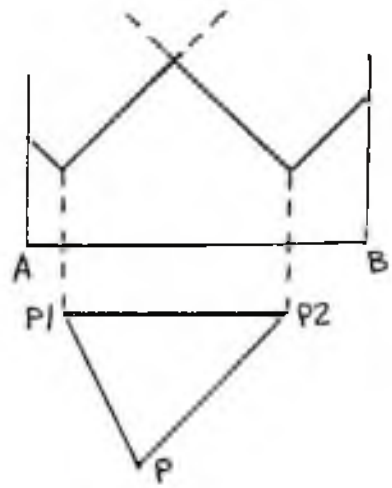


Fig. 3.5 Distancia desde un origen a una recta cuando existe un obstáculo entre el origen y la recta.

Sea que se considere la función de distancia respecto a un origen existiendo o no obstáculo entre el origen y la recta es importante destacar el hecho de que si bien existe una función de distancia la misma que puede determinar la distancia de cualquier punto respecto al origen, interesa de entre todos los puntos de la recta aquel que tiene la distancia menor respecto al punto origen. En varios casos de los aquí presentados es evidente la ubicación del punto hasta el cual la distancia es mínima, en otros se debe escoger entre un conjunto de puntos.

3.2 EL CONCEPTO DE RANURA.

Sea una recta representativa del lado de una figura. Se desea accesar a la recta desde uno o varios puntos orígenes y obtener sobre la recta, con relación a cada punto origen, tanto la función de distancia como los puntos para los cuales la distancia respecto al origen es mínima.

Como se puede ver en la Fig. 3.6 estas funciones de distancia se intersectan y traslapan si se consideran varios orígenes. Cada punto origen definirá sobre la recta un dominio tal que los puntos de la recta pertenecientes a ese dominio

estarán más cercanos al origen respectivo que a los restantes orígenes. Esta es la noción fundamental de una **ranura**.

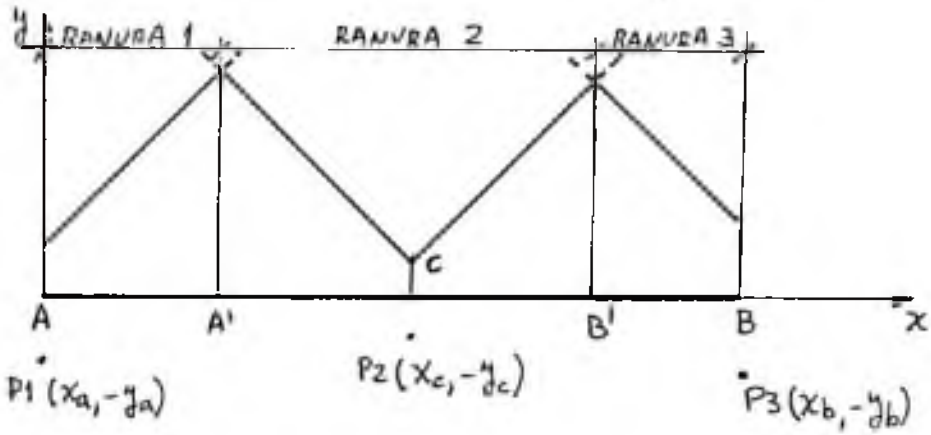


Fig 3.6 Ranuras formadas en una recta con tres orígenes.

Sin embargo, para los fines que se persiguen con este trabajo de tesis el concepto de **ranura** tiene asociado varios hechos y elementos:

a) la definición sobre la ranura de un punto o un conjunto de puntos que son los de distancia mínima desde el punto origen de la ranura.

b) el guardar este valor de distancia mínima.

c) el recordar cual es el punto origen de la ranura.

Si bien una **ranura** se define por la existencia de una función de distancia con su correspondiente dominio, lo que supone la obtención de un segmento de recta; en lo sucesivo se tomarán en cuenta solamente el o los puntos de la ranura que tienen distancia mínima con relación al origen.

Debe tenerse en cuenta estas ideas para la explicación del punto 3.4.3 relativa a la "Propagación de ranuras".

3.3 OBSERVACIONES RESPECTO A LAS RANURAS.

Todo lado de una figura puede tener dos caras, siendo cada cara la que dá a cada figura adyacente a través del lado comun. Sobre cada cara pueden existir ranuras.

El proceso de determinación de la ruta más corta entre dos puntos mediante el algoritmo que se describirá en el punto 3.4 puede ser visualizada como la obtención de una cadena de ranuras, una para cada cara de las figuras que atraviesa la ruta óptima.

Las ranuras se deben mantener conexas o convexas, es decir no debe quedar espacio en la recta sin que pertenezca a una ranura. En el apéndice B se presenta la demostración referente a

la convexidad de ranuras.

El número de componentes conexas de una ranura que se pueden generar sobre una recta es O (m^2), donde m es el número de obstáculos. Es decir, se tienen O (m^2) ranuras que llegan a un lado por los dos sentidos (por las dos caras).[3]

Sin embargo, si se toma en cuenta cada sentido separadamente y se tiene como condición necesaria la existencia de monotonicidad el número de componentes conexas se reduce a O (m) ranuras.

Por monotonicidad se entiende el hecho que si dos ranuras conexas se intersectan el resultado continua siendo conexo.

3.4 DESCRIPCION DEL METODO DE ENRUTAMIENTO.

3.4.1 INTRODUCCION.

El algoritmo es semejante al de Dijkstra visto en el capitulo 1.

Se trata de determinar la ruta de longitud minima desde un punto inicial "s" hasta un punto final "t", tratando de no intersectar los obstàculos presentes.

Se dispone de una cola prioritaria para guardar las ranuras, ordenada ascendentemente en relación a la distancia al origen.

Pasos del algoritmo:

a) Se inicia el proceso propagando la ranura que contiene a "s" hacia las figuras vecinas. Se guardan esas ranuras en la cola.

b) Si la cola no esta vacia se saca de la cola de prioridades la ranura que tiene la distancia minima al origen.
Si la cola esta vacia se termina el algoritmo con falla.

c) Se propaga la ranura hacia las figuras vecinas. La propagación de la ranura puede generar nuevas ranuras o intersectar ranuras existentes. Si son ranuras nuevas se ingresan a la cola prioritaria, si son ranuras existentes se modifican.

d) Si se culmina el proceso exitosamente se termina el algoritmo, de lo contrario se regresa a b).

Existe un caso trivial: si entre los puntos "s" y "t" no existen obstáculos la longitud de ruta puede calcularse directamente como:

- Con norma l_1:

$$d = |X_t - X_s| + |Y_t - Y_s|$$

- Con norma l_∞:

$$d = \max (|X_t - X_s| , |Y_t - Y_s|)$$

En los siguientes párrafos, del 3.4.2 hasta el 3.4.6, se explican los pasos del algoritmo.

3.4.2 INICIO DEL PROCESO.

Se inicia el proceso determinando, en la gráfica de figuras a la figura que contiene al punto inicial "s", obteniéndose para todos los lados de la figura, mediante los cuales se comunica con las figuras vecinas (lados que no son obstáculos), las funciones de distancia y por consiguiente las ranuras respectivas. Se determinan además los valores de distancia mínima desde "s" hasta puntos de referencia ubicados en las ranuras obtenidas.

Se guardan en la cola prioritaria las ranuras. La cola esta ordenada ascendentemente en relación a la distancia al origen medida sobre la ranura.

3.4.3 PROPAGACION DE RANURAS.

Un vez inicializado así el procedimiento de enrutamiento, este continua mediante una sucesión de etapas similares a las del algoritmo de Dijkstra y que efectúan la propagación de la ruta a través de las ranuras.

La ranura a propagarse es aquella para la cual se tiene la distancia mínima en el frente de la cola, asignando además a su punto de distancia mínima como origen local para la propagación.

Definida así la ranura a propagarse se efectúan la secuencia de pasos indicada en la inicialización :

a) se obtienen las funciones de distancia para cada lado de la figura a la que pertenece la ranura que se propaga, siempre que mediante estos lados exista comunicación hacia las figuras vecinas.

b) se establece el dominio de cada función, es decir se definen las nuevas ranuras.

c) se determina sobre cada ranura un punto referencial que es aquel de distancia mínima respecto al origen local.

d) se actualiza el contenido de la cola prioritaria tomando en consideración que las distancias a guardarse correspondientes a los puntos referenciales deben ser las totales desde el punto inicial "s". Es decir, si O es el origen local, "p" el punto referencial encontrado para la nueva ranura y d la distancia desde "s" hasta O, la distancia total desde "s" hasta "p" es $|Op| + d$, valor que corresponderá a d cuando a esta nueva ranura le toque propagarse [3].

Como se explicó en el punto 3.3, con el fin de mantener ranuras convexas, es decir, que en el proceso de obtención de ranuras no queden espacios sin analizarse y además para evitar que se produzcan ciclos a lo largo de sucesivas propagaciones, es decir que al propagar una determinada ranura se regrese a la ranura original de la cual provino, se asumen sentidos para la propagación de ranuras. Se tiene entonces que en un lado de una figura se pueden considerar dos caras las que corresponden al acceso a ese lado desde dos sentidos diferentes, para cada cara del lado podrán determinarse funciones de distancia y, de hecho, cada ranura que corresponda a cada cara se maneja independientemente.

La Fig. 3.7 ilustra los sentidos de propagación y las caras de los lados de las figuras. Las caras '1' corresponden al sentido 1 de propagación y las caras '2' al sentido 2 de propagación.

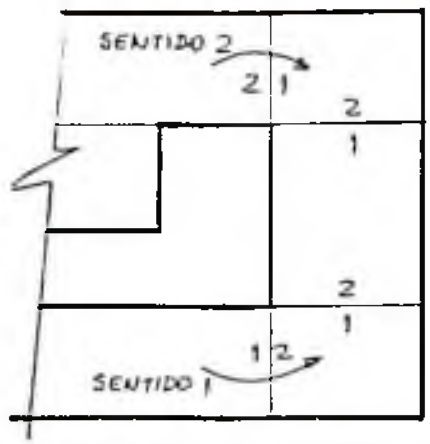


Fig. 3.7 Sentido de propagación de ranuras.

3.4.4 INTERSECCION DE RANURAS.

A lo largo de la propagación de la ruta a través de las ranuras puede presentarse el hecho de que se llegue al mismo lado de una figura y a su misma cara, por diferentes caminos, es decir desde diferentes ranuras pero mediante el mismo sentido de propagación. Esto es equivalente a tener sobre un mismo lado varias ranuras, es decir definir varias funciones de distancia en relación a diferentes puntos de origen, tal como se explicó en el punto 3.2. Debido a que cada función de distancia tiene su correspondiente dominio existirán puntos de intersección de estas funciones, esto se denomina intersección de ranuras.

La intersección se presenta en diferentes etapas de propagación de la ruta, pues en una etapa cualquiera se generará sobre el lado una sola ranura. Se supone entonces que en una determinada etapa de propagación se creó una ranura en el lado y que posteriormente en otra etapa se llegó al mismo lado, generándose otra ranura que modificará a la ranura original en el sentido que cada una de ellas cubrirá en su dominio un porcentaje del lado de la figura. Los diferentes dominios de las ranuras quedarán definidos por los puntos de intersección de las

funciones de distancia correspondientes.

En la Fig. 3.8 se tienen, por ejemplo. dos ranuras que se intersectan en un punto C. Supongamos que la primera ranura en generarse sobre el lado AB es aquella con punto origen el punto P1, esta ranura tendrá como dominio la totalidad del lado pues se trata de la única ranura generada en AB, el punto de referncia para esta ranura es el vértice A. Posteriormente se genera una segunda ranura, la que tiene como punto de origen el punto P2 y como punto de referncia el vértice B. Cada ranura tendrá como dominio un porcentaje del lado AB. Interesa determinar el valor de la abscisa del punto de intersección Xc.

Para determinar Xc se aplica la ecuación :

$$(Xc - Xa) + Ya = (Xb - Xc) + Yb \text{ , de donde}$$

$$Xc = \frac{Xa + Xb}{2} + \frac{Yb - Ya}{2}$$

es decir, desde el punto de abscisa media de AB se desplaza el valor (Yb - Ya)/2 para determinar la abscisa Xc. Una ranura tendrá como dominio el intervalo Xa <= x <= Xc y la otra tiene como dominio el intervalo Xc <= x <= Xb.

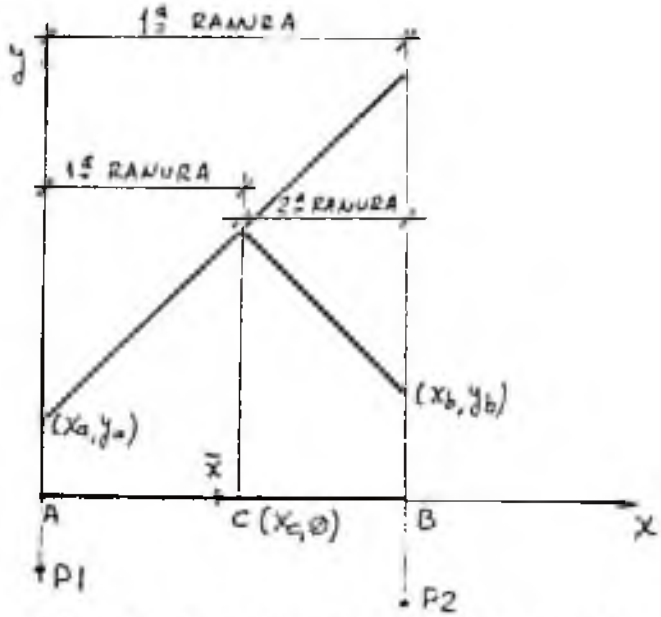


Fig. 3.8 Intersección de ranuras.

Esta situación de intersección de ranuras no se presenta, en cambio, cuando se consideran las dos caras de un lado de una figura es decir diferentes ranuras y funciones de distancia sobre el mismo lado pero desde diferentes sentidos de propagación.

Si dos ranuras obtenidas en diferentes etapas de generación

y con el mismo sentido de propagación coinciden en el mismo punto de referencia provocan la siguiente decisión:

sea d1 la distancia hasta el punto de referencia de la primera ranura generada y d2 la distancia hasta el mismo punto pero considerando la segunda ranura.

si d1 › d2 se elimina la primera ranura y queda solamente la segunda.

si d1 ‹= d2 permanece la primera ranura y la segunda no se mantiene para futuras propagaciones.

3.4.5 CULMINACION DEL PROCESO DE ENRUTAMIENTO.

El punto final de destino "t" debe pertenecer a una ranura. Si coincide con un vértice o un lado de una figura de hecho se cumple con esta condición, de otra manera será necesario efectuar una partición adicional de la figura que contiene a "t" para conseguir la condición indicada, tal como se ilustra en la Fig. 3.9.

Fig. 3.9 Partición de la figura que contiene a t, para conseguir que t pertenezca a una ranura.

Si en el proceso de propagación de ruta se llega hasta la ranura que contiene a "t" entonces la obtención de la ruta termina con éxito, calculándose la distancia hasta "t", pero el proceso de propagación de ranuras continúa y se detiene solamente cuando la distancia mínima contenida en la ranura corriente que se va a propagar es mayor que la distancia alcanzada hasta "t". Esto se comporta así debido a que pueden existir varias rutas desde "s" hasta "t".

3.4.6 PERMANENCIA DE UNA RANURA ACTIVA.

Las funciones de distancia que definen una ranura consideran un valor de distancia mínima hasta un punto de la ranura y un valor de distancia máxima hasta otro punto de ella.

Una ranura permanecerá activa, aún después de haberse propagado mientras la distancia mínima guardada y actualizada en la ranura corriente que se va a propagar sea menor que la

distancia máxima para la ranura.

En aquellos casos en los que exista intersección de ranuras es claro que la ranura adicional que modifica a una ya existente puede propagarse en su dominio aún después que la ranura original se haya propagado y siga activa; esto se debe a que, como ya se indicó, cada ranura se maneja de manera independiente. Por otro lado, si la ranura original ya no esta activa, la nueva ranura actuará en un dominio que cubrirá un mayor porcentaje (posiblemente todo el lado) y tendrá la posibilidad de propagarse de manera normal, como cualquier otra ranura.

3.5 ALGORITMO DETALLADO DE ENRUTAMIENTO.

Lo expresado en los puntos anteriores se sintetiza en los algoritmos que siguen, en los cuales, por un lado se presenta el algoritmo general y otro relativo a la generación de ranuras.

Previamente a la descripción de los algoritmos se indica, a continuación la representación utilizada para las ranuras, la misma que consta de los siguientes datos:

a) el punto de referencia dado por sus coordenadas.

b) el valor de la distancia acumulada desde el punto inicial "s" hasta el punto de referencia de la ranura.

c) un indicador del tipo de la ranura, con la misma convención introducida para el tipo de los obstáculos, en el capitulo 2.

d) dos indicadores, uno referente a la figura de la cual proviene la ranura y el segundo referente a la figura a la que pertenece la ranura. Estos indicadores se utilizan para representar el sentido de avance de la propagación.

e) un apuntador a la ranura que se propagó y generó la nueva ranura. Este apuntador se utiliza para reconstruir la ruta una vez que se llegó al punto final.

3.5.1 ALGORITMO GENERAL.

Sea el espacio R^2 en el cual se tienen m obstáculos horizontales, verticales y oblicuos. Sea n el número total de figuras geométricas en que estos obstáculos dividen al espacio R^2. Se ha visto que $n <= 3m + 1$.

Se definen "s" el punto inicial y "t" el punto final o destino, dados por sus coordenadas.

Hagamos:

COLA = NIL ,
O = s , O será el origen para ranuras.
d = 0 , d guardará la distancia total desde "s".

1. Para la figura a la que pertenece "s" se obtienen, con origen en "s" las ranuras hacia las figuras adyacentes y se guardan en COLA las mismas en un orden ascendente respecto a la distancia (dist) hasta su punto de referencia.

2. Si COLA esta vacio entonces se finaliza sin éxito la obtención de la ruta (si no se ha encontrado ninguna), si no se sigue a 3.

3. Se saca de la COLA la ranura que tenga dist minima, se hace O = Pi, d = dist y se propaga la ranura obteniéndose las nuevas ranuras.

   Se ingresan las nuevas ranuras a la COLA y se actualiza esta, es decir se reordena de manera ascendente en relación a las distancias totales desde "s" hasta los puntos de referencia de las ranuras, las cuales se calculan como dist = d + |OPi|.

   Si más de una ranura de la COLA contiene la misma dist minima para todas ellas se sigue el mismo procedimiento de propagación.

   Si una ranura no puede propagarse se vuelve a 2.

4. Si en la propagación se ha llegado a la ranura que contiene a "t" se calcula la distancia hasta t, se guarda ese valor y se pasa a 5; de lo contrario se pasa directamente a 5.

5. Si no existe valor para la distancia hasta "t" se vuelve al paso 3.

   Si existe valor para la distancia hasta "t" se compara el valor del contenido de la COLA con esa distancia, si el contenido de la COLA sigue siendo menor que la distancia hasta "t" se vuelve a 3, de lo contrario finaliza el procedimiento y se reconstruye la sucesión de puntos que determinan la ruta, mediante los apuntadores hacia atras contenidos en cada ranura propagada.

3.5.2 ALGORITMO DETALLADO DE GENERACION DE RANURAS.

Se accede a la lista de figuras vecinas de la figura que contiene a la ranura a propagarse. Para cada figura de esta lista de vecinos se aplica el siguiente algoritmo:

1. Se toman de uno en uno del lados de la figura seleccionada en la lista de vecinos. Si el lado es obstáculo se pasa al siguiente lado, de lo contrario se pasa al paso 2.

2. Se genera la ranura correspondiente sobre el lado considerado.

   Con la nueva ranura se sigue el siguiente análisis
3. Si se trata de una nueva ranura se la habilita inmediatamente y se termina el algoritmo.

4. Si existe otra ranura sobre el mismo lado pero con diferente punto de referencia se habilita la nueva ranura generada (situación de intersección de ranuras), se termina con el algoritmo.

5. Si existe otra ranura sobre el mismo lado y coinciden los puntos de referencia se analiza los valores de las distancias acumuladas hasta los puntos de referencia. Si la distancia que trae la nueva ranura es menor que la de la existente, la nueva ranura reemplaza a la existente, de lo contrario permanece la existente y no se habilita a la nueva ranura. Se termina con el algoritmo.

3.6 COMPLEJIDAD DEL ALGORITMO.

Se debe distinguir al efectuar el anàlisis del algoritmo de enrutamiento las diversas operaciones que se ejecutan en los diferentes pasos del algoritmo:

Sea **n** el número de figuras de la gràfica de figuras.

La determinación de la figura a la cual pertenece la ranura que se propaga es O (log(n)) por cada vez que se busque, pues las figuras de la gràfica estan guardadas en una estructura de árbol.

La obtención de las nuevas ranuras provenientes de una propagación tiene que ver con el recorrido de la lista de figuras vecinas y ya se vió que ese recorrido puede ser, en el peor de los casos de n/2 figuras cada vez, por lo que el algoritmo en esta parte es $O(n^2)$ en el peor de los casos pudiendo llegar a ser O (n).

La inserción de una ranura en la cola, la recuperación de la ranura de distancia mínima, y la eliminación de un elemento de la cola pueden ser efectuadas, cada una de ellas con un orden de O (log(r)) cada vez, donde **r** es el número de ranuras.

Como pueden haber $O(n^2)$ ranuras el orden global del algoritmo de enrutamiento es de:

$$O(n^2 \log(n)).$$

Los ejemplos y resultados respectivos se muestran en el capítulo 5 . Los ejemplos se resuelven tanto por el método aquí expuesto como por el método de la gráfica reticular que se expone en el siguiente capítulo.

CAPITULO 4

TRABAJOS RELACIONADOS.

Como se indicó al inicio de este trabajo existen numerosos estudios efectuados en el problema de encontrar la ruta más corta entre dos puntos, se presenta a continuación el enfoque resumido de dos planteamientos efectuados en torno a este problema.

4.1 METODO DE LA GRAFICA RETICULAR (GRID GRAPH).

Futagami,Shirakawa y Ozaki [2] introducen la estructura de gráfica reticular (grid graph) la cual consiste en dividir el espacio de estudio (una región rectangular) en intervalos de longitud unitaria tanto en el sentido vertical como horizontal formando de esta manera la estructura reticular. Además de los segmentos unitarios horizontales y verticales asi generados se consideran también segmentos oblicuos a los que, de igual forma se los supone de longitud unitaria.

En la Fig. 4.1 se muestra una gráfica reticular.

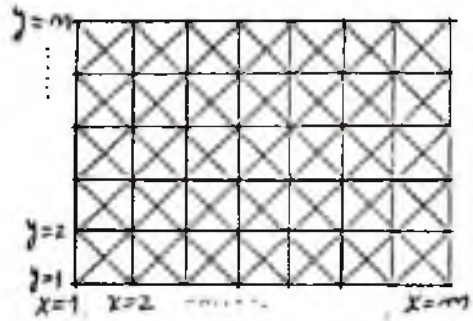


Fig. 4.1 Gráfica reticular (Grid graph).

Se consideran vértices los puntos de intersección de las líneas horizontales con las verticales exclusivamente.

Para generar una ruta determinada se avanza de vértice a vértice, sea a través de segmentos horizontales, verticales u oblicuos.

La presencia de obstáculos se representa eliminando los vértices y segmentos de cualquier clase que constituyen el obstáculo; entonces, claramente, al no existir esos vértices y segmentos la ruta debe efectuar necesariamente desvíos lo que corresponde al hecho de no intersectar al obstáculo.

4.1.1 CONCEPTOS PREVIOS.

Se introducen algunos conceptos que ayudan a una mejor comprensión de la gràfica reticular.

a) Dados dos vértices de la gráfica reticular v de coordenadas

(X_v, Y_v) y w de coordenadas (X_w, Y_w) se define :

$$dmin\ (v,w) = máx\ (|X_v - X_w|, |Y_v - Y_w|)\quad \text{donde}$$

dmin corresponde al límite inferior de la longitud de cualquier ruta existente entre v y w.

b) Dados dos vértices de la gráfica reticular v , z y un vértice w que representa cualquier vértice adyacente a v, w puede ser clasificado dentro de las siguientes tres categorías :

tipo 1 si dmin (w,z) = dmin (v,z) - 1
tipo 2 si dmin (w,z) = dmin (v,z)
tipo 3 si dmin (w,z) = dmin (v,z) + 1

En la Fig. 4.2 se muestran estas diferentes categorías.

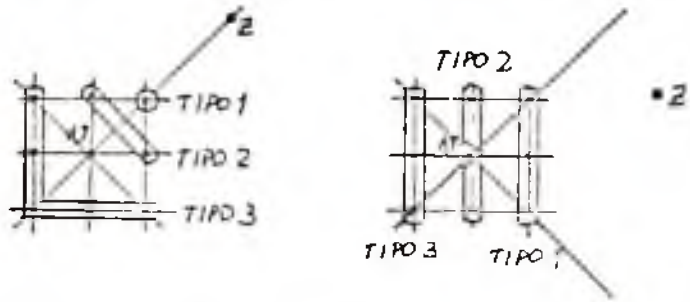


Fig. 4.2 Ejemplos de vértices adyacentes de tipo 1, 2, 3

c) Sea $p\ (s,t) = [s = v_0, v_1, v_2, \ldots, v_k = t]$ una ruta que empieza en el vértice $s = v_0$, pasa a través de otros vértices intermedios $v_1, v_2, \ldots, v_{k-1}$ y termina en el vértice $t = v_k$. Se definen los siguientes conjuntos:

TIPO 2 (p (s,t)) = {v_h (1 <= h <= k) | v_h es un vértice adyacente de tipo 2 de v_{h-1} con respecto a t}

TIPO 3 (p (s,t)) = {v_h (1 <= h <= k) | v_h es un vértice adyacente de tipo 3 de v_{h-1} con respecto a t}

Con el uso de estos conjuntos se define la longitud de desvío de p (s,t) así:

DESVIO (p (s,t)) = |TIPO 2 (p (s,t))| + 2 |TIPO 3 (p (s,t))|

donde en las expresiones en |.| se indica el número de elementos de los conjuntos.

Finalmente la longitud de la ruta mínima entre s y t se calcula con la expresión:

long (p (s,t)) = dmin (s,t) + DESVIO (p (s,t))

4.1.2 ALGORITMO DE ENRUTAMIENTO.

En base a los conceptos vertidos anteriormente se plantea el siguiente algoritmo para obtener la ruta más corta entre un punto inicial s y un punto final t.

Se disponen de tres stacks N1, N2 y N3 para guardar los vértices de tipos 1,2 y 3, respectivamente.

0. Dada una gráfica reticular G, sean s y t los puntos inicial y final. Sea D una variable entera que contiene la distancia de desvío. Sean u y v dos variables que contienen datos de vértices de la gráfica reticular.
   Se inicializa v = s;
   D = 0;
   N1 = N2 = N3 = NULL;

1. Se asigna al vértice v una etiqueta (v,z). Si v = t se pasa al paso 5, de otra manera para cada vértice w adyacente a v y que no esta etiquetado se sigue el siguiente proceso:

   - si w es un vértice adyacente de v respecto a t del tipo 1, entonces se ingresa al stack N1 el par [w,(w,v)], donde (w,v) es el lado de la gráfica reticular entre w y v.

   - si w es un vértice adyacente de v respecto a t del tipo 2, entonces se ingresa al stack N2 el par [w,(w,v)], donde (w,v) es el lado de la gráfica reticular entre w y v.

   - si w es un vértice adyacente de v respecto a t del tipo 3, entonces se ingresa al stack N3 el par [w,(w,v)], donde (w,v) es el lado de la gráfica reticular entre w y v.

2. Si N1 esta vacío se pasa al paso 3. De otra manera sea [u,(u,z)] el elemento del tope de N1. Si u esta etiquetado se saca al elemento [u,(u,z)] del stack N1 y se repite el paso 2. Si u no esta etiquetado se asigna v = u, se saca fuera del stack N1 al elemento [u,(u,z)] y se regresa al paso 1.

3. Si los restantes stacks N2 y N3 estan también vacíos se pasa al paso 4. De otra manera se transfiere el contenido del stack N2 al stack N1 y el contenido del stack N3 al N2, quedando el stack N3 vacío. Se incrementa la longitud de desvío en 1 (D = D + 1) y se regresa al paso 2.

4. Se suspende la búsqueda pues significa que no existe ruta entre los puntos s y t.

5. Se concluye la búsqueda de la ruta con éxito y se calcula la longitud de la ruta con la expresión:

$$L_{min}(s,t) = d_{min}(s,t) + D$$

   Se reconstruye la trayectoria de los vértices de la ruta, desde s hasta t a partir de las etiquetas asignadas a los

vértices.

Este método se ha utilizado como comparación del algoritmo de propagación de ranuras propuesto en este trabajo de tesis.

4.2 METODO DE LA GRAFICA DE PISTAS (TRACK GRAPH).

Wu, Widmayer, Schlag y Wong [1] introducen en cambio la estructura de la gráfica de pistas (track graph). Los obstáculos corresponden a polígonos convexos en x,y, es decir son regiones planares.

Una gráfica de pistas se define como TG (OB,N) donde :

OB = conjunto de polígonos que corresponden a los obstáculos
N = conjunto de puntos en el espacio R^2- OB, que corresponde al espacio que queda disponible para el enrutamiento.

El objetivo es obtener, a través de los puntos de N la ruta de longitud mínima para unir dos puntos del plano.

La gráfica de pistas (track graph) se construye obteniendo para cada polígono de OB el menor rectángulo que contenga al polígono, es decir prolongando e intersectando los lados extremos izquierdo, derecho, superior e inferior del polígono obtenemos este rectángulo; las aristas del rectángulo prolongadas, sea hasta el final del espacio R^2, sea hasta que se topen con el contorno de otro obstáculo corresponden a las pistas (tracks) del polígono.

En la Fig. 4.3 se muestra una gráfica de pistas.

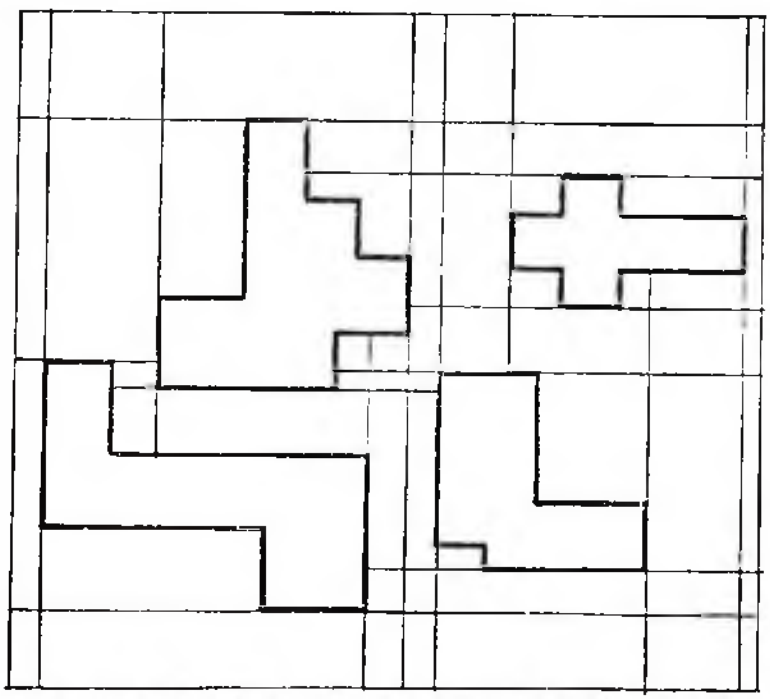


Fig. 4.3 Gráfica de pistas (track graph).

Relacionando la estructura de gráficas G[V,E] con la gráfica de pistas, cada punto de intersección entre pistas y los vértices de los obstáculos constituyen los vértices V de G y los lados E estan constituidos por pistas que unen dos vértices V o por lados del polígono o por combinaciones de pistas y lados.

Para cada uno de los puntos p del conjunto N se obtienen tambien sus pistas de punto (point tracks) hacia la izquierda, hacia la derecha, hacia arriba y hacia abajo de p. Estas pistas de punto se determinan trazando líneas (en los cuatro sentidos) desde p hasta que encuentren una pista de polígono o un lado de polígono. Obviamente si p coincide con una pista de polígono o con un lado de polígono el número de sus pistas de punto disminuyen.

En el procedimiento para determinar la ruta de longitud mínima se sigue bastante de cerca el algoritmo de Dijkstra. Se tiene un punto Po inicial. Existe un conjunto S que contiene los vértices de la gráfica de pistas ya visitadas en la búsqueda. Inicialmente S = 0. Una cola prioritaria T guarda elementos con el formato (t,d,s) donte t es un punto candidato a ser incluido en S, d es la distancia desde Po hasta t y s es el antecesor de t en la ruta desde Po hasta t. Los elementos se encuentran ordenados en la cola T en base a los valores de d. Inicialmente T tiene un solo elemento (Po,0,Po).

El procedimiento de búsqueda es así:

Se toma el elemento del tope de T (t,d,s) y lo eliminamos de T. Se incluye t en S y se registra la distancia d como su distancia y s como su antecesor. Se consideran los nodos adyacentes a t; sea v un nodo adyacente de t, si esta ya en T se compara la distancia d' guardada en el elemento de T (v,d',s') con la distancia d + d(t,v) que es la distancia de v a través de t, si d + d(t,v) es menor que d' entonces se actualiza el elemento de T como (v,d + d(t,v),t). Si v no está en T se inserta en T el nuevo elemento (v,d + d(t,v),t).

Se continúa con el proceso hasta que se haya llegado al punto destino.

CAPITULO 5

EJEMPLOS Y RESULTADOS.

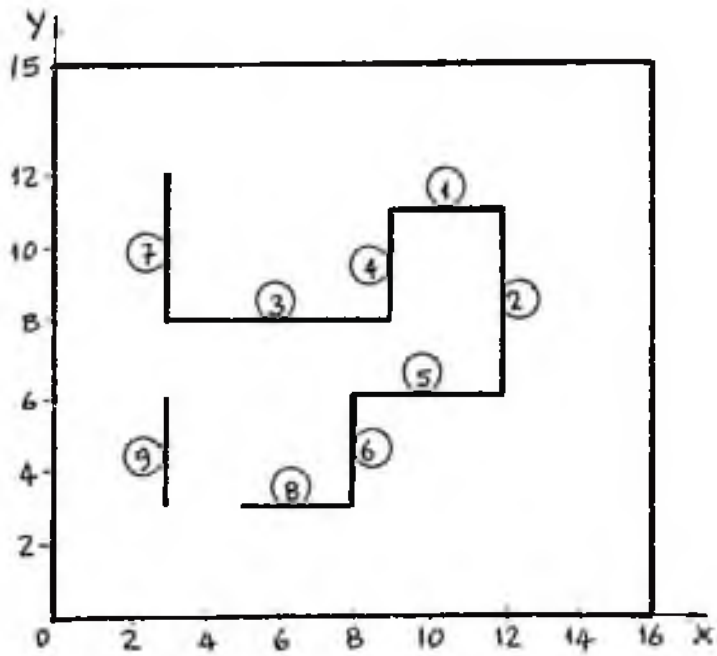
En este capitulo se presentan algunos ejemplos resueltos con aplicación a los algoritmos vistos en los capitulos precedentes. a partir de lo que los resultados expresan se formulan varias conclusiones iniciales.

Todos los programas con los cuales se han procesado los ejemplos se han redactado en el lenguaje C, utilizando el compilador Lattice C, versión 3.0 para microcomputadores personales tipo IBM PC o compatibles.

La información básica referente a los programas se indica en el apéndice C y los listados de los programas fuente en el apéndice D de este trabajo.

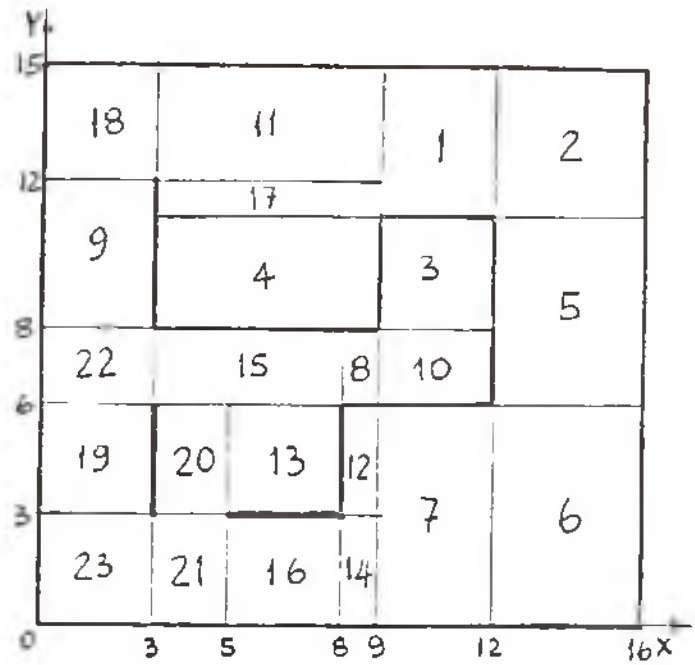
5.1 EJEMPLOS.

En los ejemplos que se indican en el resto de este capitulo se van a procesar dos planos con sus respectivos obstáculos, en lo sucesivo se denominarán Plano 1 y Plano 2. En las Fig. 5.1 y 5.2 se muestra a los dos planos, con indicación de la partición en figuras (gráfica de figuras) para la aplicación del método de la propagación de ranuras.



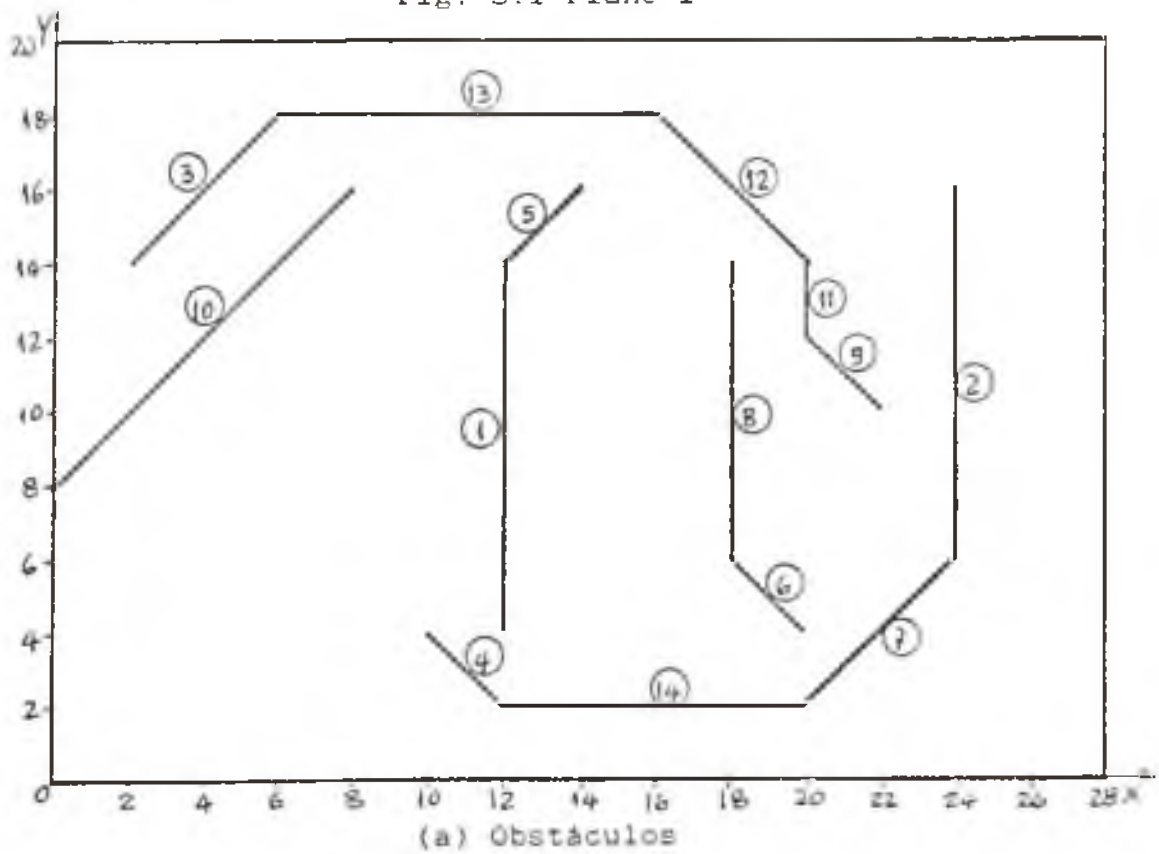
(a) Obstáculos

Fig. 5.1 Plano 1



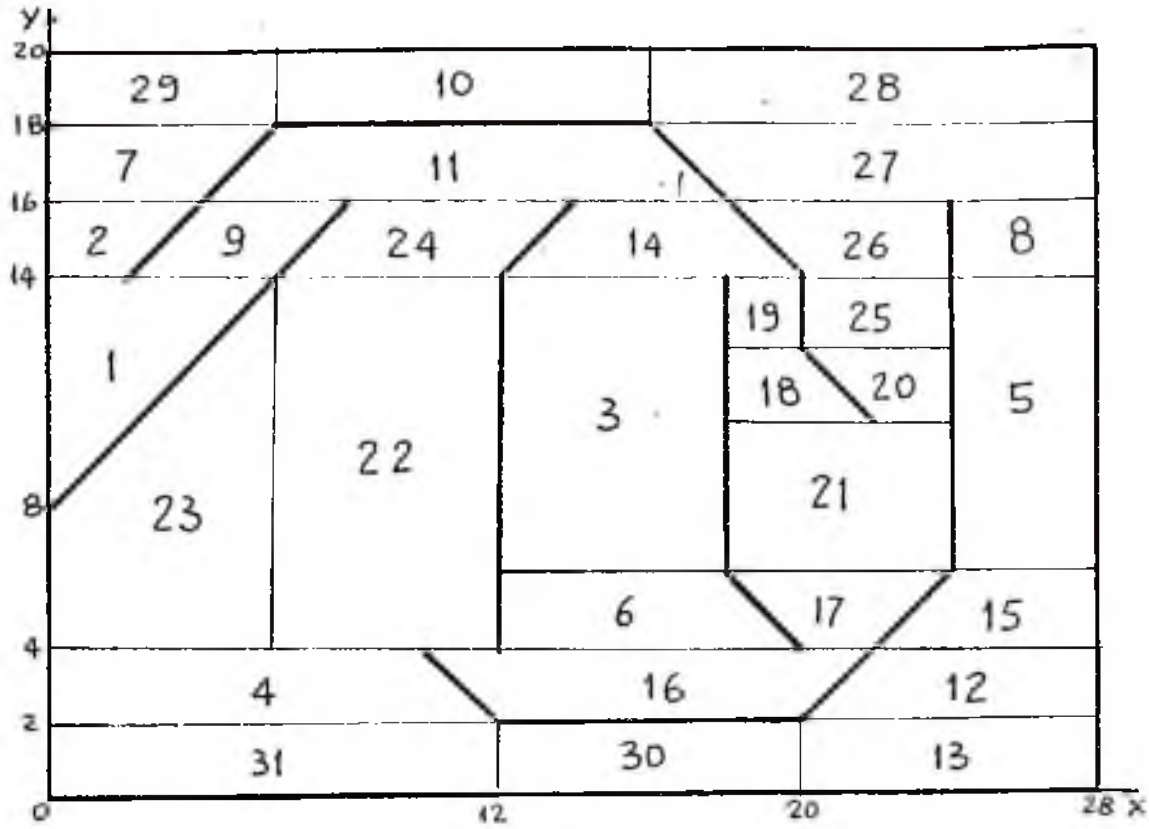
(b) Gráfica de figuras

Fig. 5.1 Plano 1



(a) Obstáculos

Fig. 5.2 Plano 2



(b) Gráfica de figuras

Fig. 5.2 Plano 2

Sobre el Plano 1 se van a encontrar las trayectorias de ruta mínima para dos pares de puntos inicial y final y sobre el Plano 2 se van a determinar las treyectorias de ruta mínima para tres pares de puntos inicial y final.

Tanto para los puntos inicial y final del Plano 1 como para los del Plano 2 las trayectorias de ruta mínima se van a determinar para dos métodos de enrutamiento:

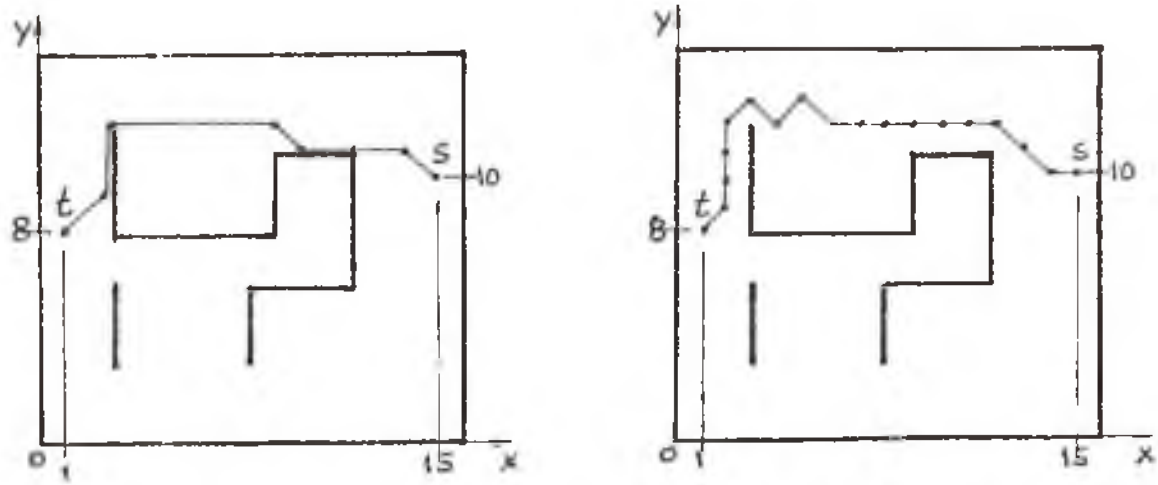
a) el método de la propagación de ranuras al que se llamará simplemente RANURAS en lo sucesivo y,

b) el método de la gráfica reticular al que se llamará RETICULA en lo que sigue.

A continuación, en la Fig. 5.3 se muestran las gráficas de las trayectorias entre los diferentes puntos inicial y final. En todos los casos se ha calculado la distancia con la norma l_∞ para el método de RANURAS con el fin de comparar el valor de distancia con el que se obtiene con el método RETICULA.

Los enrutamientos a obtenerse son los siguientes:

1) Plano 1 - Punto inicial s (15,10) Punto final t (1,8)

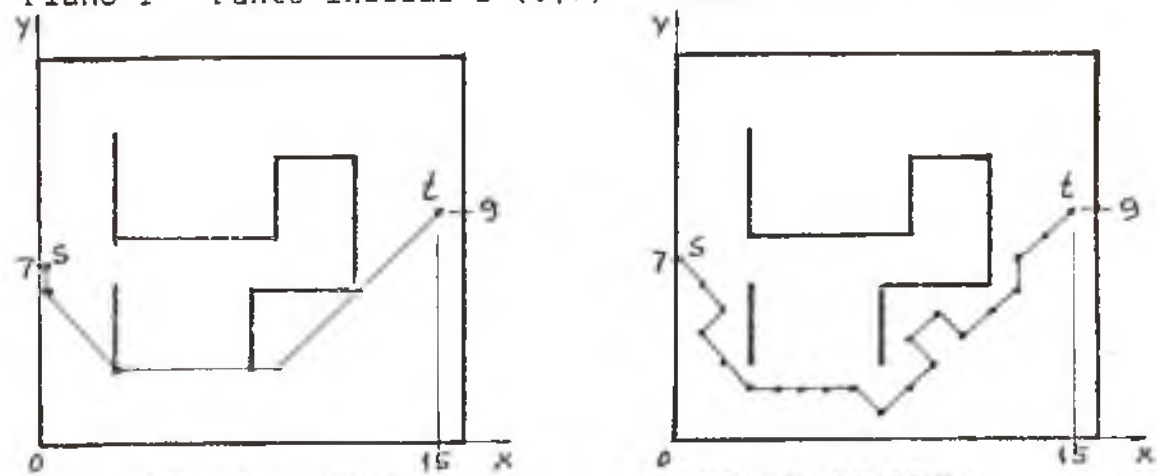


Método RANURAS

Método RETICULA

Fig. 5.3 (a)

2) Plano 1 - Punto inicial s (0,7) Punto final t (15.9)

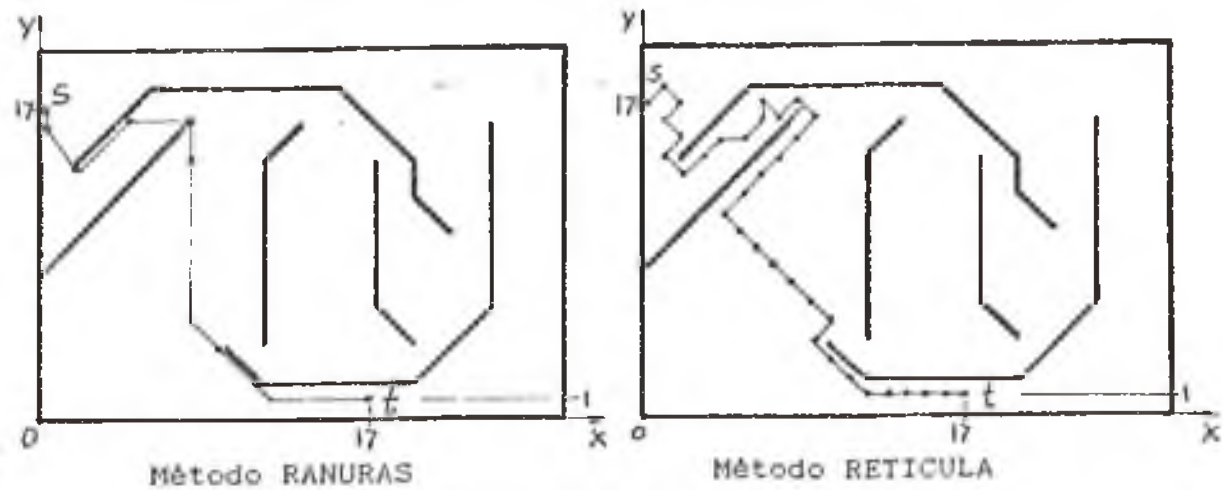


Método RANURAS

Método RETICULA

Fig. 5.3 (b)

3) Plano 2 - Punto inicial s (0,17) Punto final t (17,1)

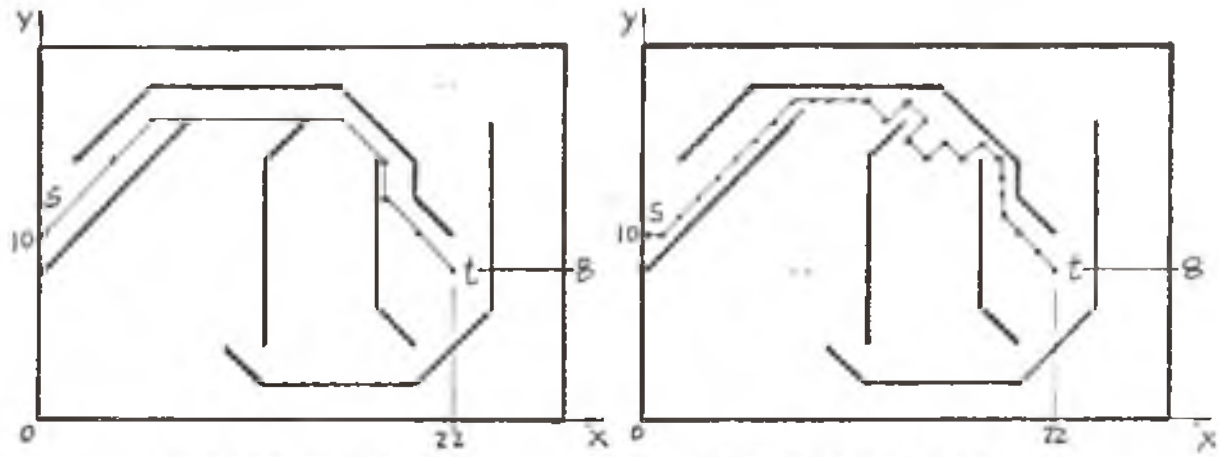


Método RANURAS

Método RETICULA

Fig. 5.3 (c)

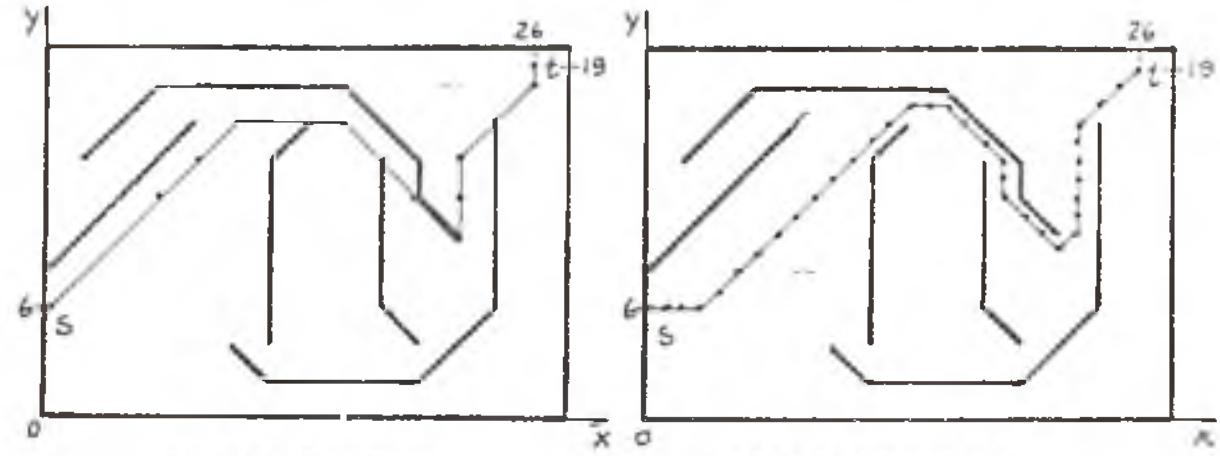
4) Plano 2 - Punto inicial s (0,10) Punto final t (22,8)



Método RANURAS Método RETICULA

Fig. 5.3 (d)

5) Plano 2 - Punto inicial (0,6) Punto final (26,19)



Método RANURAS Método RETICULA

Fig. 5.3 (e)

Se presentan a continuación los resultados más relevantes de las rutas anteriormente mostradas, en forma tabular:

Plano 1.

Número de obstáculos = 9 Número de figuras = 23

PUNTOS		METODO							
		RANURAS				RETICULA			
s	t	N.R.P.	N.R.R.	Tie.	Dis.	N.P.V.	N.P.R.	Tie.	Dis.
15,10	1,8	20	7	8	16	84	18	3	16
0,8	15,9	30	8	12	16	60	21	1	17

Plano 2.

Número de obstáculos = 14 Número de figuras = 31

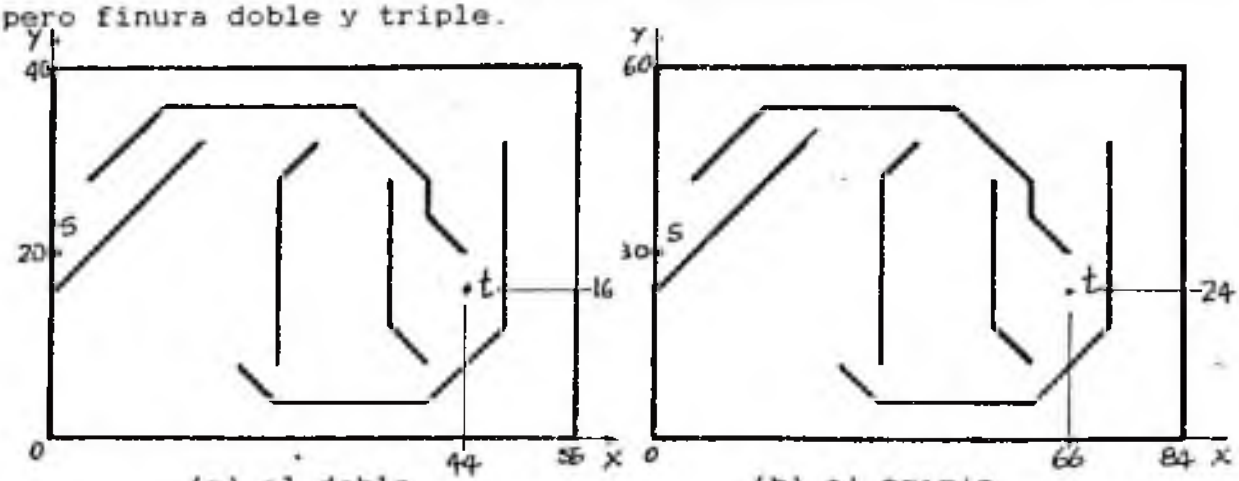
PUNTOS		METODO							
		RANURAS				RETICULA			
s	t	N.R.P.	N.R.R.	Tie.	Dis.	N.P.V.	N.P.R.	Tie.	Dis.
0,17	17,1	28	9	9	28	150	35	6	29
0,10	22,8	24	8	5	24	83	28	3	24
0,6	26,19	36	13	14	31	256	35	20	34

Abreviaturas:

N.R.P. = número de ranuras propagadas,
N.R.R. = número de ranuras en la ruta óptima,
N.P.V. = número de puntos visitados,
N.P.R. = número de puntos en la ruta óptima,
Dis. = distancia en unidades de longitud,
Tie. = tiempo en segundos.

Como puede verse estos primeros resultados no son muy alentadores pues apenas si en uno de las cinco procesos de enrutamiento el tiempo utilizado por el método de propagación de ranuras es menor que el utilizado por el de la gráfica reticular, sin embargo se puede ver un rendimiento mejor del método de propagación de ranuras en lo que al cálculo de la distancia se refiere.

Con el fin de tener más elementos de comparación se decidió verificar qué pasa cuando se cambia la finura, es decir la longitud del intervalo en la reticula. En la Fig. 5.4 se muestran los nuevos planos de estudio que corresponden al mismo Plano 2 pero finura doble y triple.



(a) al doble (b) al triple

Fig. 5.4 Plano 2 modificado

Se procesaron con los dos métodos, de propagación de ranuras y de la gráfica reticular los siguientes enrutamientos:

a) En el Plano 2 modificado al doble en su grado de finura:
   Punto inicial s (0,20) Punto final t (44,16)

b) En el Plano 2 modificado al triple en su grado de finura:
   Punto inicial s (0,30) Punto final t (66,24)

Los resultados de estos procesamientos se muestran en la siguiente tabla:

PUNTOS		METODO							
		RANURAS				RETICULA			
s	t	N.R.P.	N.R.R.	Tie.	Dis.	N.P.V.	N.P.R.	Tie.	Dis.
0,20	44,16	24	8	5	48	288	50	20	48
0,30	66,24	24	8	5	72	685	74	112	72

Donde se puede ver la influencia del grado de finura para el método de la gráfica reticular, en tanto que este no afecta al método de la propagación de ranuras.

5.2 ANALISIS DE LOS RESULTADOS.

Se han presentado algunos ejemplos de obtención de trayectorias entre diferentes puntos de diferentes planos. También se han obtenido trayectorias entre dos puntos para el caso de planos proporcionales (a diferente escala) que representan diferente grado de finura de la retícula cuando de aplica el método de la gráfica reticular.

Los rsultados muestran los siguientes hechos :

a) Se tiene entera similitud tanto en la forma de la trayectoria como en el valor de la misma para los dos métodos utilizados. Sin embargo se observa que en algunos casos de la aplicación del método de la gráfica reticular se obtienen valores algo mayores para la distancia calculada entre los mismos puntos con el método de la propagación de ranuras. Esto se debe a que el método de la gráfica reticular puede presentar eventualmente algún recorrido innecesario.

b) El método de la propagación de ranuras es dependiente del número de figuras que se tienen en la gráfica de figuras y ese número es a su vez dependiente del número de obstáculos existentes. Se recuerda que el número de figuras es 3n + 1, donde n es el número de obstáculos. En los ejemplos 3 y 4 se tiene siempre el mismo número de figuras y debido a eso el número de ranuras propagadas en planos proporcionales y uniendo puntos igualmente proporcionales es constante.

c) El método de la gráfica reticular es en cambio dependiente del grado de finura de la retícula. En efecto, en los ejemplos 3, y 4 el grado de finura de la retícula pasó de 1 a 2 y a 3. El hecho de que el grado de finura de la retícula afecta a las dos direcciones de medida del plano sugiere que la función del número de nodos visitados en el método de la gráfica reticular es una función cuadrática.

En efecto, supongamos que la función es:

$$f(n) = an^2 + bn + c,$$

donde n es el grado de finura.

En base al número de nodos visitados para los diferentes grados de finura se plantea el siguiente sistema de ecuaciones:

para n = 1 $\quad 83 = a + b + c$

para n = 2 $\quad 288 = 4a + 2b + c$

para n = 3 $\quad 685 = 9a + 3b + c$

Resolviendo el sistema de ecuaciones se tiene:

$$f(n) = 96n^2 - 83n + 70$$

d) Posiblemente el inconveniente del método de la propagación de ranuras es el tamaño del archivo GRAFICA.DAT, pues cada nodo tiene asociada una lista de figuras que son los vecinos. Sin embargo, hay un hecho alentador, conforme aumenta el número de obstáculos se van "dibujando" en el plano ciertas rutas y eso hace que el número de vecinos para una figura cualquiera tienda a disminuir.

CONCLUSIONES.

En la parte final del capitulo 5 se han presentado ya algunas conclusiones derivadas de los resultados obtenidos en los ejemplos. Adicionalmente y en referencia al método de la gráfica de figuras se concluye lo siguiente:

El método que se ha presentado, basado en la gráfica de figuras trata sobre todo de simplificar la gráfica de búsqueda. El número de vértices de la gráfica (O (m) donde m es el número de obstáculos)no es tan grande como en otros métodos como el de la gráfica reticular y la búsqueda de la ruta se puede efectuar de manera más directa.

El método de la propagación de ranuras es más general que el de la gráfica de pistas pues este esta definido solamente para la norma l_1 y obstáculos rectangulares.

En el método de la gráfica de pistas los puntos de pista (point tracks) vienen a ser lo que en el método propuesto en este trabajo son las ranuras. El método de la gráfica de pistas genera a todos los puntos de pista , en el método de la propagación de ranuras estas se generan conforme se necesiten.

COMENTARIOS.

Los tipos de figuras que se obtienen en la gráfica de figuras vienen dados por la aplicación en sí, por esta razón se generan cuadriláteros y triángulos. Debido a esto no se aplicó a la división del plano un método de triangulación que posiblemente contemple una complejidad menor que la alcanzada en este trabajo.

Algunos aspectos no tratados en esta tesis y sobre los cuales se puede continuar trabajando son:

En este trabajo se ha aplicado una metodología incremental para construir figuras basada en la adición de obstáculos. Sin embargo, la metodología incremental no es tan eficiente como la de la construcción de la gráfica de figuras en un solo paso.

Otro aspecto constituye la detección en la gráfica de figuras de conectividades y puntos de articulación , esto permitirá que no se efectúen recorridos innecesarios en la gráfica a través de nodos que no conducen a la obtención de la ruta.

REFERENCIAS Y BIBLIOGRAFIA.

[1] Y.F.Wu, P. Widmayer, M.D.F. Schlag, C.K. Wong
"Rectilinear Shortest Paths and Minimun Spanning Trees in the Presence of Ractilinear Obstacles" IEEE Transactions on Computers. Vol C-36 No.3, March 1987. pp 321-331.

[2] S.Futagami, J. Shirikawa, H. Ozaki.
"An Automatic Routing Sistem for single-layer Printed Wiring Boards". IEEE Transactions on Circuits and Systems, Vol CAS-29, No.1, January 1982.pp. 46-51.

[3] D.M. Mount
"Voronoi Diagrams on the Surface of a Polyhedron" Tech. Rept 121, Center for Automation Research, University of Maryland, (May 1985).

[4] C.Y. Lee
"An algorithm for path connections and its applications". IRE Trans. Electron. Comp., vol EC-10, pp 346-365, Sept 1961.

[5] T. Lozano Pérez
"Automated planning of manipulator transfer movements". IEEE Trans. Syst., Man, Cybern, vol SMC-11, pp 681-698, Oct 1981.

[6] A. Guttman
"R-Trees: A dynamic index structure for spatial searching". ACM SIGMOD, Proc. SIGMOD 84, Vol 14, No 2, June 1984.

[7] H. Samet
"The quadtree and related hierarchical data structures". Computing Surveys Vol 16, No 2, June 1984.

[8] R.E. Tarjan
"Data Structures and Networks Algorithms". Society for Industrial and Applied Mathematics, 1983.

[9] N. Deo
"Graph Theory with Applications to Engineering and Computer Science" Prentice-Hall, Inc. 1974.

[10] A. Aho, J. Hopcroft, J. Ullman
"The Design and Analysis of Computer Algorithms". Addison-Wesley Publishing Co. 1974.

APENDICE A.

FUNCIONES DE DISTANCIA PARA LA NORMA l_∞.

De la misma manera como existen las funciones de distancia desde un punto origen a los puntos de una recta con la norma l_1 se puede efectuar lo propio con la norma l_∞.

Para esto se tiene una recta AB y un punto origen P, referidos a un sistema de coordenadas (x,y).

Se trata de determinar la función de distancia de los puntos de la recta AB en relación con el punto origen P.

La función de distancia variará conforme se recorra la recta a lo largo de su eje desde el un extremo hasta el otro y dependerá en su expresión algebraica de la posición de P respecto a los extremos de la recta.

Los tipos de funciones que pueden considerarse son los ya indicados para la norma l_1 (funciones tipo f1 a f7).

1.1 LA RECTA ES HORIZONTAL O VERTICAL.

Se pueden presentar todos los tipos de funciones definidos, aunque en el caso del tipo f5, este corresponderá a una situación muy particular. Los valores de las pendientes para las funciones f1 y f2 serán de -1 y +1, respectivamente.

Para este tipo de rectas se pueden analizar los siguientes casos:

1.1.1 El punto origen se ubica fuera del segmento AB.

En este caso se tienen las siguientes posibilidades:

1) La distancia con norma l_∞ es $|Yr - Yp|$, es decir la mayor distancia esta en dirección Y.

Se tienen las funciones:

a) Para el intervalo $[Xa,(|Yr - Yp| - |Xa - Xp|)]$

$$\text{función} = |Yr - Yp|$$

Recta paralela al eje x.

b) Para el intervalo $[(|Yr - Yp| - |Xa - Xp|),Xb]$

$$\text{función} = x + |Xa - Xp|$$

Recta a 45 grados respecto al eje x.

Estas funciones se ilustran en la figura A.1.

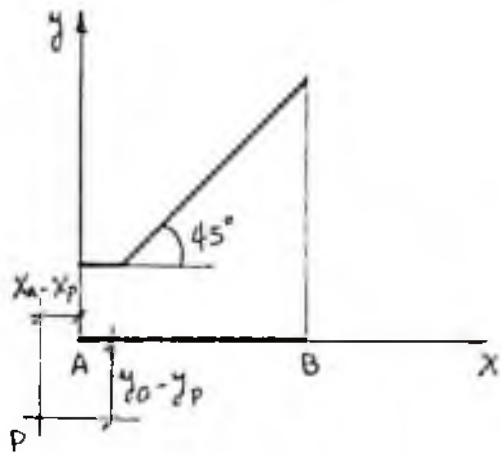


Fig. A.1 Funciones de distancia con norma l_∞, el origen esta fuera del intervalo [Xa,Xb], distancia máxima = |Yr - Yp|

2) La mayor distancia es en dirección X, es decir, la distancia con norma l_∞ es |Xa - Xp|.

Se tiene una sola función de distancia para todo el intervalo [Xa,Xb]:

función = x + |Xa - Xp|

Esta función se ilustra en la figura A.2.

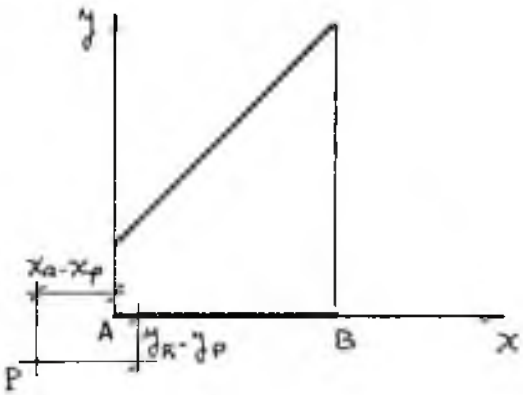


Fig. A.2 Función de distancia para norma l_∞, el origen esta fuera del intervalo [Xa,Xb], distancia máxima = |Xa - Xp|.

Se puede observar que si P es coincidente con la dirección AB pero Xp < Xa la función de distancia anteriormente descrita se aplica sin ninguna modificación.

Expresiones enteramente análogas pueden detenerse para los casos en los cuales la abscisa del punto P es mayor que la del extremo B.

1.1.2 El punto origen se ubica entre los extremos de la recta AB.

Para este caso la distancia desde P hasta la recta estará dada siempre por la diferencia de ordenadas |Yr - Yp|, la función de distancia contempla en general tres funciones asi:

a) Para el intervalo [Xa,(Xp - |Yr - Yp|)]

$$\text{función} = -x + Xp$$

Recta a 135 grados con el eje x.

b) Para el intervalo [(Xp - |Yr - Yp|),(Xp + |Yr - Yp|)]

$$\text{función} = |Yr - Yp|$$

Recta paralela al eje x.

c) Para el intervalo [(Xp + |Yr - Yp|),Xb]

$$\text{función} = x + |Xb - Xp|$$

En la situación particular en la cual la distancia |Yr - Yp| es cero el punto P pertenecerá a la recta AB y se elimina la segunda función Y = |Yr - Yp| quedando solo las dos restantes las mismas que tendrán su punto de intersección en P. Para esta condición se tiene un tipo de función f5.

Las funciones de distancia anteriormente descritas se muestran en la figura A.3.

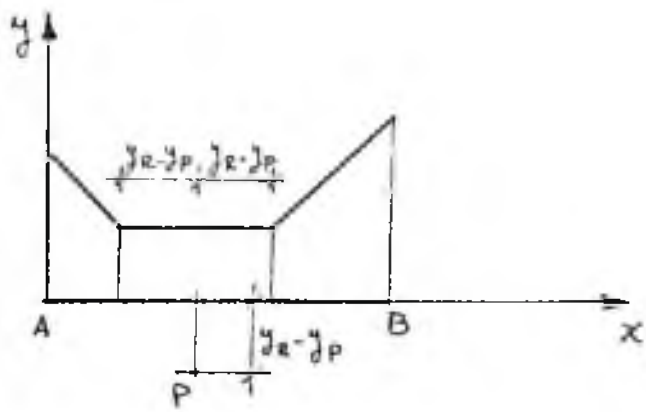


Fig A.3 Funciones de distancia con norma l_∞, el origen se encuentra en el intervalo [Xa,Xb]

1.2 LA RECTA ES OBLICUA.

Los tipos de función que se obtienen en este análisis son f1, f2 y f5. En el caso de las funciones f1 y f2 los valores de las pendientes son -1 y +1, respectivamente.

La descripción de las diferentes funciones de distancia se

efectuará en base a una recta a 45 grados y con el punto origen ubicado bajo la recta.

Se analizan los siguientes casos, en los cuales interesa la variación de las coordenadas del punto origen, tanto en dirección x como en dirección y:

1) La ordenada del origen es menor que la del punto inferior izquierdo de la recta. Para esta situación la abscisa del origen puede variar así:

a) Es menor que la abscisa del punto inferior izquierdo (Xp < Xa)
Para el intervalo [Xa,Xb]

función = x + max (|Ya - Yp| , |Xa - Xp|)

b) Esta comprendida entre las dos abscisas de los extremos de la recta (Xa <= Xp <= Xb), o
es mayor que la abscisa del extremo superior derecho de la recta (Xp > Xb).

Si |Yp - Ya| >= |Xp - Xa| se tiene una función igual a la del caso a), es decir:

función = x + |Yp - Ya|

Si |Yp - Ya| < |Xp - Xa| se tiene:

Para el intervalo [Xa,Xa + (|Xp-Xa| - |Yp-Ya|)/2]

función = |Xa - Xp| - x

Para el intervalo [Xa + (|Xp-Xa| - |Yp-Ya|)/2,Xb]

función = x + |Ya - Yp|

2) La ordenada del origen esta comprendida entre las ordenadas de los puntos extremos de la recta. Para esta situación la abscisa del origen varia así:

Esta comprendida entre las abscisas de los puntos extremos de la recta (Xa <= Xp <= Xb), o
es mayor que la abscisa del extremo superior derecho de la recta (Xp > Xb).

Para el intervalo
[Xa,Xa + |Yp-Ya| + (|Xp-Xa| - |Yp-Ya|)/2]

función = |Xa - Xp| - x

Para el intervalo
[Xa + |Yp-Ya| + (|Xp-Xa| - |Yp-Ya|)/2,Xb]

función = x + |Ya - Yp|

3) La ordenada del origen es mayor que la del extremo superior derecho de la recta. Para esta situación la abscisa del origen varía así:
Para el intervalo [Xa,Xb]

$$función = max (|Yp - Ya| , |Xp - Xa|) - x$$

La Fig. A.4 ilustra las funciones de distancia analizadas anteriormente.

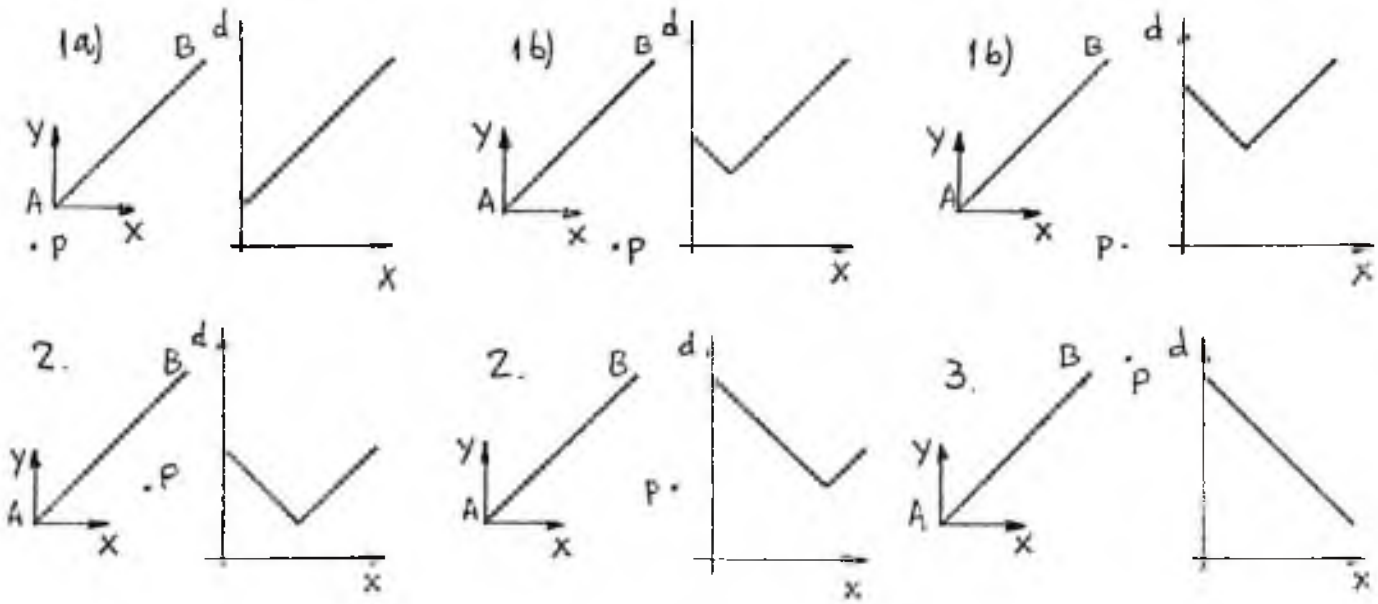


Fig. A.4 Funciones de distancia para la norma l_∞, desde un origen a una recta oblicua.

Análisis similares pueden efectuarse para el caso de una recta oblicua a 45 grados, con el punto origen ubicado sobre la recta o para el caso de una recta oblicua a 135 grados.

Finalmente, tal como se indicó en la descripción de las funciones de distancia para la norma l_1, es necesario ubicar al punto para el cual se tiene el valor mínimo de la distancia, aunque, como se puede ver en varias funciones, puede existir todo un segmento de recta (un conjunto de puntos) para el cual se tiene este valor mínimo de distancia. Aún en ese caso debe individualizarse a un solo punto, de entre todos los del segmento.

APENDICE B.

CONVEXIDAD DE RANURAS.

Las ranuras generadas sobre cada lado de una figura son convexas, es decir cubren la totalidad del lado; esto lleva al planteamiento del siguiente teorema :

TEOREMA:

el lugar geométrico de los puntos de una ranura equidistante a dos orígenes diferentes es una cadena monotónica.

PRUEBA:

sean dos puntos orígenes, se asume por comodidad el uno en el origen de coordenadas (0,0) y el segundo en un punto de coordenadas (X1, Y1). Se asume además que la ruta generada hasta el origen (0,0) tiene un cierto peso d. Desde estos dos orígenes se accede a una misma ranura. Los puntos de la ranura equidistantes a los dos orígenes cumpliran con la ecuación :

$$|X - X1| + |Y - Y1| = |X| + |Y| + d \qquad (1)$$

Debido a los términos con valores absolutos de la ecuación (1) pueden presentarse varias formas de ella, dependiendo del signo que toman las expresiones contenidasn en los valores absolutos.

Se analizan los siguientes casos:

1. $X - X1 > 0 \Rightarrow X > X1$, bajo esta condición Y puede variar de la siguiente manera :

1.1 $Y - Y1 > 0 \Rightarrow Y > Y1$, la ecuación queda:

$$X - X1 + Y - Y1 = X + Y + d$$

$$-X1 - Y1 = d,$$

Esta expresión no representa ninguna solución real y válida, de lo que se concluye que no existe solución para la ecuación (1) en esa región del plano.

1.2 $Y - Y1 < 0$, $Y > 0 \Rightarrow 0 < Y < Y1$, la ecuación queda:

$$X - X1 - Y + Y1 = X + Y + d$$

$$Y = \frac{Y1 - X1 - d}{2}$$

Se tendría en esta región del plano una línea recta paralela al eje X, sin embargo Y será positiva siempre que se cumpla la condición $Y1 > X1 + d$, condición que no se puede afirmar se

cumplirá siempre, en consecuencia, en esta región del plano no existe solución a la ecuación (1).

1.3 $Y - Y1 < 0$, $Y < 0 \Rightarrow Y < 0$, la ecuación queda :

$$X - X1 - Y + Y1 = X - Y + d$$

$$d = Y1 - X1,$$

Esta expresión no tiene significado práctico pues es simplemente una relación entre tres números que son constantes en el problema que se esta tratando, en consecuencia en esta región del plano no existe solución a la ecuación (1).

2. $X - X1 < 0$, $X > 0 \Rightarrow 0 < X < X1$, para esta condición Y puede variar de la siguiente manera :

2.1 $Y - Y1 > 0 \Rightarrow Y > Y1$, la ecuación queda:

$$-X + X1 + Y - Y1 = X + Y + d$$

$$X = \frac{X1 - Y1 - d}{2}$$

Solución similar a la del punto 1.2, se tendria en esta región del plano una recta paralela al eje Y, sin embargo, debido a que la condición para X es que sea positiva esta se cumplirá siempre que $X1 > Y1 + d$, condición que no se puede generalizar, en consecuencia en esta región se considera que no existe solución a la ecuación (1).

2.2 $Y - Y1 < 0$, $Y > 0$, $\Rightarrow 0 < Y < Y1$, la ecuación queda :

$$-X + X1 - Y + Y1 = X + Y + d$$

$$X + Y = \frac{X1 + Y1 - d}{2} \qquad (2)$$

La ecuación de esta recta constituye solución válida a la ecuación (1) para esta región del plano considerada.

2.3 $Y - Y1 < 0$, $Y < 0$, $\Rightarrow Y < 0$, la ecuación queda:

$$-X + X1 - Y + Y1 = X - Y + d$$

$$X = \frac{X1 + Y1 - d}{2} \qquad (3)$$

Se obtiene para esta región del plano una solución válida que corresponde a una recta paralela al eje Y.

3. $X < 0$, para esta condición Y varía de la siguiente manera :

3.1 $Y - Y1 > 0, \Rightarrow Y > Y1$, la ecuación queda:

$$- X + X1 + Y - Y1 = - X + Y + d$$

$$d = X1 - Y1$$

Al igual que en el numeral 1.3 esta expresión no tiene ningun significado práctico pues representa una relación entre tres constantes del problema, en consecuencia en esta región del plano no existe solución a la ecuación (1).

3.2 $Y - Y1 < 0$, $Y > 0, \Rightarrow 0 < Y < Y1$, la ecuación queda:

$$- X + X1 - Y + Y1 = - X + Y + d$$

$$Y = \frac{X1 + Y1 - d}{2} \qquad (4)$$

Se obtiene para esta región del plano una solución válida que corresponde a una recta paralela al eje X.

3.3 $Y - Y1 < 0$, $Y < 0$, $\Rightarrow Y < 0$, la ecuación queda:

$$- X + X1 - Y + Y1 = - X - Y + d$$

$$d = X1 + Y1$$

Nuevamente esta expresión corresponde a una relación entre las constantes del problema y al no tener significado práctico se concluye que no existe solución a la ecuación (1) en esta región del plano.

Para probar la continuidad de la cadena monotónica, se observa que al reemplazar en la ecuación de la recta obtenida en el numeral 2.2 los valores de $X = 0$, $Y = 0$ se obtienen las expresiones de las rectas obtenidas en los numerales 3.2 y 2.3, respectivamente :

La expresión obtenida en 2.2 es :

$$Y = - X + \frac{X1 + Y1 - d}{2} \qquad (2)$$

Para $X = 0$

$$Y = \frac{X1 + Y1 - d}{2} \qquad (4)$$

expresión alcanzada en 3.2

Para Y = 0

$$X = \frac{X1 + Y1 - d}{2} \qquad (3)$$

expresión alcanzada en 2.3

Las soluciones a la ecuación (1) y las regiones del plano analizadas se muestran en la figura B-1.

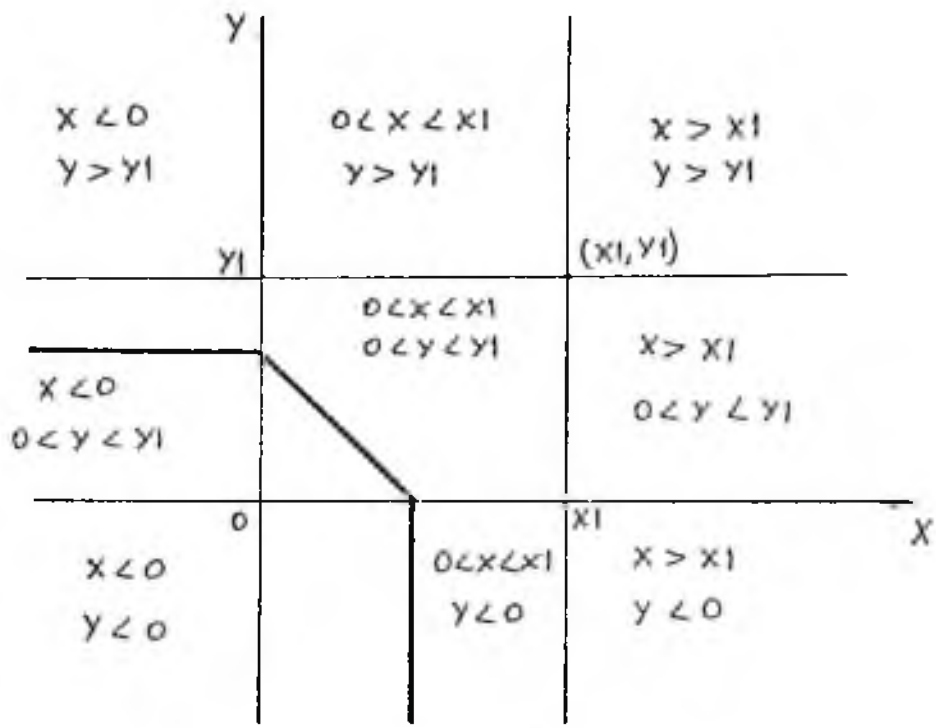


Fig. B-1 Lugar geométrico de los puntos de una ranura equidistantes a dos orígenes diferentes.

APENDICE C.

DESCRIPCION DE LOS PROGRAMAS DESARROLLADOS.

Como se indicó en el capitulo 5 todos los programas se han desarrolado en el lenguaje C, compilador Lattice C, version 3.0.

1. ESTRUCTURAS DE DATOS GENERALES.

En varios capitulos precedentes se han definido la representación de varios objetos como obstàculos, figuras, ranuras, etc. que se manejan en este trabajo. Las estructuras de datos generales correspondientes estan acordes con esas definiciones. En el apèndice D se indican las estructuras utilizadas tanto para el método de la propagación de ranuras como para el de la gráfica reticular.

Adicionalmente, los obstáculos se almacenan internamente en un arreglo de estructuras del tipo segmento.

El programa que maneja el método de la gráfica reticular utiliza una matriz de enteros de tamaño máximo 100x100 para representar al plano.

2. PROGRAMAS Y SU UTILIZACION.

2.1 METODO DE GRAFICA DE FIGURAS - PROPAGACION DE RANURAS.

Se tienen dos programas que se pueden manejar de manera independiente en este método, el mismo que constituye el principal objetivo de este trabajo de tesis.

2.1.1 PROGRAMA GRAFICA.

Este programa tiene dos formas de utilización:

a) Para generar una nueva gráfica, en este caso se deben ingresar las coordenadas de los límites del plano de estudio, al cual se lo supone rectangular y los datos de los obstáculos referentes a las coordenadas de los puntos inicial y final. Este ingreso de datos se efectúa a través del teclado, conforme el programa lo vaya solicitando. Una vez que se hayan ingresado los datos del último obstáculo el programa crea dos archivos de datos para ser utilizados en otros programas:

2.1.1.1 ARCHIVO GRAFICA.DAT

En el archivo GRAFICA.DAT cada figura se guarda en una línea, la misma que tiene el siguiente formato:

col 1-5 : indicador con la siguiente convención:
1 es la figura básica del nodo.
0 es una figura de la lista de vecinos.
col 6-10 : número de identificación de la figura.
col 11-15 : indicador del tipo de figura.

col 16-20 : indicador de la posición de la figura.
col 21-25,26-30,31-35,36-40 : indicadores para cada lado de la figura de si es o no obstáculo.
col 41-45,46-50,51-55,56-60,61-65,66-70,71-75,76-80 : valores de las coordenadas de los vértices de la figura, en el orden que se indicó en el capitulo 2.

2.1.1.2 ARCHIVO OBSTAC.DAT

El archivo OBSTAC.DAT guarda los datos de los obstáculos, los contornos del plano de estudio son los primeros obstáculos y luego se almacenan los restantes. Cada obstáculo se guarda en una línea con el siguiente formato :

col 1-5,6-10,11-15,16-20 : coordenadas de los extremos del obstáculo.
col 21-25 : indicador del tipo de obstáculo.

Los resultados de la gráfica de figuras, mostrados en un tipo de salida accesible al usuario, se despliegan en la pantalla una vez concluido el procesamiento del último obstáculo y se guardan también en el archivo FIGURA.DAT.

b) Para modificar una gráfica ya existente. En este caso el programa añade nuevos obstáculos a una gráfica que ya existía previamente. El programa lee los datos constantes en los archivos GRAFICA.DAT y OBSTAC.DAT e inmediatamente se ingresan los datos de los nuevos obstáculos conforme el programa lo solicite. Una vez introducido el último obstáculo la gráfica modificada se guarda en el archivo GRAFICA.DAT, los obstáculos. añadidos los nuevos se guardan en el archivo OBSTAC.DAT y los resultados finales para el usuario de la gráfica de despliegan en la pantalla y se guardan tambie en el archivo FIGURA.DAT.

Respecto a las características del programa GRAFICA, este consta de aproximadamente 6000 líneas de código fuente y su programa ejecutable ocupa un espacio de 84 KB.

2.1.2 PROGRAMA PROPAGA.

Este programa es el que obtiene la ruta de longitud mínima entre dos puntos utilizando el método de la propagación de ranuras.

El programa lee inicialmente los datos de la gráfica constantes en el archivo GRAFICA.DAT, inmediatamente se debe suministrar la opción de cálculo de la distancia (con norma l_1 o con norma l_∞) y finalmente se ingresan los datos de las coordenadas de los puntos inicial y final, entre los cuales se desea encontrar la ruta de longitud mínima.

El resultado de la ejecución del programa se despliega en la pantalla y también se guarda en el archivo TRAYEC.DAT.

El programa PROPAGA consta de aproximadamentre 1500 líneas

de código y su programa ejecutable ocupa una extensión de 34 KB.

2.2 METODO DE LA GRAFICA RETICULAR - PROGRAMA GRAFRET.

Este programa obtiene la ruta de longitud mínima entre dos puntos del plano con el método de la gráfica reticular.

El programa inicializa la matriz representativa del plano en 0, lee y utiliza el archivo de obstáculos OBSTAC.DAT colocando un -1 en los elementos de la matriz que coinciden con los obstáculos, inmediatamente se ingresan los datos de las coordenadas de los puntos inicial y final como datos adicionales para la obtención de la ruta. El resultado se despliega en la pantalla y se guarda también en el archivo TRAYEC.DAT.

El programa GRAFRET consta de aproximadamente 900 líneas de código fuente y ocupa una extensión de 24 KB.

APENDICE D

PROGRAMAS

ESTRUCTURAS DE DATOS - PROGRAMAS GRAFICA Y PROPAGA

```
#define NUMPAR 5     /* numero maximo de particiones de una figura */
#define NUMVER 4     /* numero maximo de vertices de una figura */
#define NUMPT 2      /* numero de puntos de un segmento */
#define MAXOBS 100   /* numero maximo de obstaculos */

struct punto {
   int x;
   int y;
   };

struct segmento {
   struct punto inicial;
   struct punto final;
   int tipo;            /* tipo de segmento, horizontal, vertical, oblicuo */
   };

struct cuadrilatero {
   struct punto infizq;
   struct punto supizq;
   struct punto supder;
   struct punto infder;
   int indob [NUMVER];  /* indicador de lados obstaculos */
   int figura;          /* tipo de figura, cuadrilatero o triangulo */
   int pos;             /* posicion de la figura, normal o rotada */
   int ident;           /* numero de identificacion de la figura */
   };

struct ranura {
   struct punto refer;
   int rtipo;
   int dista;
   int idfig;
   int desde;
   struct ranura *previo;
   };

struct lstcua {          /* lista ligada de figuras vecinas */
   struct cuadrilatero conten;
   struct lstcua *prox;
   };

struct elemento {                  /* elemento de la grafica */
   struct cuadrilatero basico;     /* la figura en si */
   struct lstcua *vecino;           /* lista de figuras vecinas */
   };

struct lstcdr {          /* lista ligada de figuras */
   struct elemento nodo;
   struct lstcdr *next; };
```

```
truct queue {           /* lista ligada prioritaria de ranuras */
   struct ranura info;
   struct queue *sigui;
   };

truct cerrado {         /* lista ligada de ranuras propagadas */
   struct ranura visit;
   struct cerrado *proxi;
   };

truct pila {             /* estructura de pila para desplegar la ruta */
   struct ranura infor;
   struct pila *sigu;
   };
```

RCHIVO DE FUNCIONES EXTERN - PROGRAMAS GRAFICA Y PROPAGA

```
xtern struct lstcdr *donde(), *lstfig(), *actual(), *continua(), *borra();
xtern struct lstcdr *ubica(), *busca(), *leegra(), *listar();
xtern struct lstcua *formlst(), *nuevec(), *lstrec();
xtern struct queue *disvec(), *inserta(), *recupera();
xtern constrans(), consrot(), invierte(), muestra(), ensena(), imprime();
xtern escribe(), cdrlro(), triglo(), desco1(), desco2(), formapt(), obveci();
xtern selcuat(), seltres(), obtipo(), figpos(), rotdir(), rotinv();
xtern compara(), copelem(), copranu(), comprue(), par(), atravi(), divide();
xtern obshor(), obsver(), obli45(), obli135(), rtdirs(), rtinvs(), figpos();
xtern dive20(), dive21(), dive22(), dive30(), dive31(), dive32(), dive33();
xtern dive34(), dive35(), dive36(), dive37(), dive38(), dive40(), dive41();
xtern dive42(), dive43(), dive44(), dive45(), dive46(), dive47(), dive48();
xtern dive49(), dive49a(),dive50(), dive51(), dive52();
xtern char *malloc();
xtern free();
```

```
/* PROGRAMA GRAFICA.

#include "include\stdio.h"
#include "estruc.h"
#include "externo.h"

int xmin,xmax,ymin,ymax,nob = 0;
int abin,abfi,orin,orfi;
struct cuadrilatero parte [NUMPAR];

/* programa para generar las particiones producidas por un obstaculo */

main()

    FILE *fpg, *fps, *fpf, *fopen();
    struct punto p1,p2,p3,p4,pin,pfi;
    struct segmento obst [MAXOBS];
    struct lstcdr *ap,*apt1,*apt2, *base = NULL;
    struct lstcua *pv;
    struct cuadrilatero rct;
    int nob = 0,si,id0,id1,id2,id3,i,nb,ca,sigue,opcion,fclose();

    printf ("\nGENERACION DE UNA GRAFICA A PARTIR DE UN PLANO\n");
    printf ("INICIAL Y OBSTACULOS QUE LO PARTICIONAN O\n");
    printf ("MODIFICACION DE UNA GRAFICA YA CONSTRUIDA ANADIENDO\n");
    printf ("NUEVOS OBSTACULOS\n");
    printf ("\nELIGE UNA OPCION :\n");
    printf ("\n1. Generacion de una nueva grafica.\n");
    printf ("2. Modificacion de una grafica ya existente.\n");
    printf ("\nOPCION : ") ; scanf ("%d",&opcion);
    while ((opcion != 1) && (opcion != 2)) {
        printf ("Opcion no valida. Por favor ingrese nuevamente.\n");
        printf ("OPCION : ") ; scanf ("%d",&opcion); }
    switch (opcion) {
    case 1 : {
        printf ("\nGENERACION DE UNA NUEVA GRAFICA\n");
        datgen ();
        formapt (&p1,xmin,ymin) ; formapt (&p2,xmin,ymax);
        formapt (&p3,xmax,ymax) ; formapt (&p4,xmax,ymin);
        for (i = 0; i < NUMVER; i++)
            switch (i) {
            case 0 :
                lstobs (&obst[i],&p1,&p2);
                break;
            case 1 :
                lstobs (&obst[i],&p2,&p3);
                break;
            case 2 :
                lstobs (&obst[i],&p4,&p3);
                break;
            case 3 :
                lstobs (&obst[i],&p1,&p4);
                break;
            }
        nb = NUMVER - 1;
```

```
    } break;
case 2 : {
    printf ("\nMODIFICACION DE UNA GRAFICA EXISTENTE\n");
    fpg = fopen ("grafica.dat","r");
    base = leegra (base,fpg);
    fclose (fpg);
    fps = fopen ("obstac.dat";"r");
    i = 0;
    while ((ca = leeobs (&obst[i].fps)) != EOF)
        i++;
    nb = --i;
    fclose (fps);
    printf ("\nDATOS DE LA GRAFICA CARGADOS\n");
    } break;
}
printf ("\nDesea impresion de la grafica despues");
printf (" de cada obstaculo ? (s/n) ");
getchar() ; si = getchar() ; sigue = 's';
printf ("\nDATOS DE LOS OBSTACULOS\n") ; printf ("\n");
while ((sigue == 's') || (sigue == 'S')) {
    lectura(++nob);
    formapt (&pin,abin,orin) ; formapt (&pfi,abfi,orfi);
    lstobs (&obst[++nb],&pin,&pfi);
    apt1 = donde (base,&obst[nb],&id1,&id2);
    id0 = id1;
    if (id2 == 1) {
        ap = apt1->next;
        apt2 = donde (ap,&obst[nb],&id1,&id3);
    }
    if (apt1 == NULL) {
        cdrlro (&rot,&p1,&p2,&p3,&p4);
        for (i = 0; i <NUMVER; i++)
            rot.indob[i] = 1;
        divide (&rot,&obst[nb]);
        base = lstfig (base,apt1);
    }
    else {
        divide (&apt1->nodo.basico,&obst[nb]);
        base = lstfig (base,apt1);
        base = actual (base,apt1);
        base = borra (base,apt1);
    }
    if (id0 == -1) {
        pv = apt1->nodo.vecino;
        while (pv != NULL) {
            id3 = atravi (&pv->conten,&obst[nb]);
            if ((id3 == 0) || (id3 == -1)) {
               apt1 = ubica (base,pv->conten.ident);
               divide (&apt1->nodo.basico,&obst[nb]);
               base = lstfig (base,apt1);
               base = actual (base,apt1);
               base = borra (base,apt1);
               while (id3 == -1) {
                  apt1 = continua (base,&obst[nb],apt1->nodo.vecino,&id3);
                  divide (&apt1->nodo.basico,&obst[nb]);
```

```
                    base = lstfig (base,apt1);
                    base = actual (base,apt1);
                    base = borra (base,apt1);
                }
            }
            pv = pv->prox;
        }
    }
    if (id2 == 1) {
        divide (&apt2->nodo.basico,&obst[nb]);
        base = lstfig (base,apt2);
        base = actual (base,apt2);
        base = borra (base,apt2);
        if (id1 == -1) {
            pv = apt2->nodo.vecino;
            while (pv != NULL) {
                id3 = atravi (&pv->conten,&obst[nb]);
                if ((id3 == 0) || (id3 == -1)) {
                    apt2 = ubica (base,pv->conten.ident);
                    divide (&apt2->nodo.basico,&obst[nb]);
                    base = lstfig (base,apt2);
                    base = actual (base,apt2);
                    base = borra (base,apt2);
                    while (id3 == -1) {
                        apt2 = continua (base,&obst[nb],apt2->nodo.vecino,&id
                        divide (&apt2->nodo.basico,&obst[nb]);
                        base = lstfig (base,apt2);
                        base = actual (base,apt2);
                        base = borra (base,apt2);
                    }
                }
                pv = pv->prox;
            }
        }
    }
    if ((si == 's') || (si == 'S'))
        desplie (base);
    printf ("Existe otro obstaculo ? (s/n) ") ;
    getchar() ; sigue = getchar();
}
if ((si != 's') && (si != 'S'))
    desplie (base);
fpf = fopen ("figura.dat","w");
muestra (fpf,base);
fclose (fpf);
fpg = fopen ("grafica.dat","w");
graba (base,fpg);
fclose (fpg);
fps = fopen ("obstac.dat","w");
for (i = 0; i <= nb; i++)
    guarda (&obst[i],fps);
fclose (fps);
printf ("\nGRAFICA GENERADA\n");
```

```
* funcion que ingresa los limites del plano de estudio */
atgen ()

    printf ("\nLIMITES DEL PLANO DE ESTUDIO\n");
    printf ("\nAbscisa izquierda = ");
    scanf ("%d",&xmin);                          */
    printf ("Abscisa derecha = ");
    scanf ("%d",&xmax);
    printf ("Ordenada inferior = ");
    scanf ("%d",&ymin);
    printf ("Ordenada superior = ");
    scanf ("%d",&ymax);


/* funcion que ingresa los datos de un obstaculo */
ectura (i)
nt i;

    printf ("\nObstaculo N %d\n",i);
    printf ("\nCoordenadas del punto inicial del obstaculo :\n");
    printf ("\nAbscisa :");
    scanf ("%d",&abin);
    printf ("Ordenada :");
    scanf ("%d",&orin);
    printf ("\nCoordenadas del punto final del obstaculo :\n");
    printf ("\nAbscisa :");
    scanf ("%d",&abfi) ;
    printf ("Ordenada :");
    scanf ("%d",&orfi);


* funcion que forma el arreglo de obstaculos dados los puntos */
stobs (s,pt1,pt2)
truct segmento *s;
truct punto *pt1, *pt2;

    s->inicial.x = pt1->x ; s->inicial.y = pt1->y;
    s->final.x = pt2->x ; s->final.y = pt2->y;
    if (pt1->y == pt2->y)
        s->tipo = 0;
    else
        if (pt1->x == pt2->x)
            s->tipo = 1;
    else
        if ((pt2->y - pt1->y) * (pt2->x - pt1->x) > 0)
            s->tipo = 2;
    else
        s->tipo = 3;


* funcion que guarda en archivo el arreglo de obstaculos */
uarda (ls,ar)
truct segmento *ls;
ILE *ar;
```

```
    fprintf (ar,"%5o%5d",ls->inicial.x,ls->inicial.y);
    fprintf (ar,"%5d%5d",ls->final.x,ls->final.y);
    fprintf (ar,"%5d\n",ls->tipo);


" funcion que lee el arreglo de obstaculos */
eeobs (ob,ar)
truct segmento *ob;
ILE *ar;

    int c;

    c = fscanf (ar,"%5d%5d",&ob->inicial.x,&ob->inicial.y);
    if (c == EOF)
        return (c);
    else {
        fscanf (ar,"%5d%5d",&ob->final.x,&ob->final.y);
        fscanf (ar,"%5d\n",&ob->tipo);
    }
    return (c);


* funcion que establece las regiones en las que se divide un cuadrilatero *
ivide (clr,s)
truct cuadrilatero *clr;
truct segmento *s;

    switch (s->tipo) {
    case 0 :
        obshor (clr,s);
        break;
    case 1 :
        obsver (clr,s);
        break;
    case 2 :
        obbli45 (clr,s);
        break;
    case 3 :
        obbli135 (clr,s);
        break;
    }
    ladobst (clr,s);


* funcion que determina cuales lados de una figura son obstaculos */
adobst (cdro,sg)
truct cuadrilatero *cdro;
truct segmento *sg;

   struct punto p1,p2;
   int i,j,ti;

   for (i = 0; i < NUMPAR; i++)
       if (parte[i].ident != -1)
           switch (parte[i].figura) {
```

```
        case 0 : case 1 : case 2 : case 3 : case 4 : case 5 :
            for (j = 0; j < NUMVER; j++) {
                selcuat (&parte[i],&p1,&p2,j);
                obtipo (&p1,&p2,&ti);
                parte[i].indob [j] = compseg (cdro,sg,&p1,&p2,ti);
            }
            break;
        case 6 : case 7 : {
            parte[i].indob [3] = 0;
            for (j = 0; j < NUMVER - 1; j++) {
                seltres (&parte[i],&p1,&p2,j);
                obtipo (&p1,&p2,&ti);
                parte[i].indob [j] = compseg (cdro,sg,&p1,&p2,ti);
            }
            } break;
        }


* funcion que recorre la lista de obstaculos determinando si un lado de una */
* figura corresponde al total o parte de un obstaculo */
ompseg (cdr,s,pta,ptb,td)
truct cuadrilatero *cdr;
truct segmento *s;
truct punto *pta,*ptb;
nt td;

   struct punto pin,pfi;
   int ind = 0,tb,i;

   switch (cdr->figura) {
   case 0 : case 1 : case 2 : case 3 : case 4 : case 5 :
       for (i = 0; ((i < NUMVER) && (ind == 0)); i++)
           if (cdr->indob[i] == 1) {
               selcuat (cdr,&pin,&pfi,i);
               obtipo (&pin,&pfi,&tb);
               ind = esobst (&pin,&pfi,pta,ptb,tb,td);
           }
       break;
   case 6 : case 7 :
       for (i = 0; ((i < NUMVER - 1) && (ind == 0)); i++)
           if (cdr->indob[i] == 1) {
               seltres (cdr,&pin,&pfi,i);
               obtipo (&pin,&pfi,&tb);
               ind = esobst (&pin,&pfi,pta,ptb,tb,td);
           }
       break;
   }
   if (ind == 0) {
       desco2 (s,&pin,&pfi,&tb);
       ind = esobst (&pin,&pfi,pta,ptb,tb,td);
   }
   return (ind);


* funcion auxiliar a compseg que establece si un lado de una */
```

```
/* figura es o no obstaculo */
esobst (pi,pf,pa,pb,to,tl)
struct punto *pi,*pf,*pa,*pb;
int to,tl;
{
    int da,db,res;

    if (to == tl)
        switch (to) {
        case 0 :
            if ((pa->y == pi->y) && (pa->x >= pi->x) && (pb->x <= pf->x))
                res = 1;
            else
                res = 0;
            break;
        case 1 :
            if ((pa->x == pi->x) && (pa->y >= pi->y) && (pb->y <= pf->y))
                res = 1;
            else
                res = 0;
            break;
        case 2 : {
            da = pa->y - pa->x + pi->x - pi->y;
            db = pb->y - pb->x + pi->x - pi->y;
            if ((da == 0) && (db == 0) && (pa->x >= pi->x) && (pb->x <= pf->x))
                res = 1;
            else
                res = 0;
            }
            break;
        case 3 : {
            da = pa->y + pa->x - pi->x - pi->y;
            db = pb->y + pb->x - pi->x - pi->y;
            if ((da == 0) && (db == 0) && (pa->y >= pi->y) && (pb->y <= pf->y))
                res = 1;
            else
                res = 0;
            }
            break;
        }
    else
        res = 0;
    return (res);


/* funcion que recorre la lista general de figuras almacenando su contenido */
graba (ls,ar)
struct lstcdr *ls;
FILE *ar;

    struct lstcdr *p;

    p = ls;
    while (p != NULL) {
        grabas (&p->nodo.basico,ar,1);
```

```
        gravec (p->nodo.vecino,ar);
        p = p->next;


   funcion que recorre la lista de figuras vecinas guardando su contenido */
gravec (ls,fl)
struct lstcua *ls;
ILE *fl;

    struct lstcua *p;

    p = ls;
    while (p != NULL) {
        grabas (&p->conten,fl,0);
        p = p->prox;
    }


* funcion que imprime los datos de un cuadrilatero de la grafica */
rabas (rec,ar,ind)
truct cuadrilatero *rec;
ILE *ar;
nt ind;

    int fg,i;

    fg = rec->figura;
    fprintf (ar,"%5d%5d%5d%5d",ind,rec->ident,fg,rec->pos);
    for (i = 0; i < NUMVER; i++)
        fprintf (ar,"%5d",rec->indob[i]);
    fprintf (ar,"%5d%5d",rec->infizq.x,rec->infizq.y);
    fprintf (ar,"%5d%5d",rec->supizq.x,rec->supizq.y);
    switch (fg) {
    case 0 : case 1 : case 2 : case 3 : case 4 : case 5 :
        fprintf (ar,"%5d%5d",rec->supder.x,rec->supder.y);
        break;
    case 6 : case 7 :
        break;
    }
    fprintf (ar,"%5d%5d\n",rec->infder.x,rec->infder.y);


* funcion que recorre la lista general de figuras imprimiendo su contenido */
esplie (ls)
truct lstcdr *ls;

    struct lstcdr *p;

    p = ls;
    while (p != NULL) {
        figbas (&p->nodo.basico,1);
        figvec (p->nodo.vecino);
        getchar();
        p = p->next;
```

```
}

/* funcion que recorre la lista de figuras vecinas imprimiendo su contenido */
igvec (ls)
struct lstcua *ls;

{
   struct lstcua *p;

   printf ("\nLISTA DE FIGURAS VECINAS\n");
   printf ("\nFIGURAS N ");
   p = ls;
   while (p != NULL) {
       printf ("%5d",p->conten.ident);
       p = p->prox;
   }
   printf ("\n");


* funcion que imprime los datos de un cuadrilatero de la grafica */
igbas (rec,ind)
truct cuadrilatero *rec;
nt ind;

{
   int fg,i;

   fg = rec->figura;
   printf ("\nFIGURA DE LA GRAFICA N ") ; printf ("%d\n",rec->ident);
   printf ("\nPUNTOS\n");
   if ((fg == 6) || (fg == 7))
       printf ("Punto 1 = ");
   else
       printf ("Punto 1 (Inferior izquierdo) = ");
   printf ("%5d",rec->infizq.x);
   printf ("%5d\n",rec->infizq.y);
   if ((fg == 6) || (fg == 7))
       printf ("Punto 2 = ");
   else
       printf ("Punto 2 (Superior izquierdo) = ");
   printf ("%5d",rec->supizq.x);
   printf ("%5d\n",rec->supizq.y);
   if ((fg != 6) && (fg != 7)) {
       printf ("Punto 3 (Superior derecho) = ");
       printf ("%5d",rec->supder.x);
       printf ("%5d\n",rec->supder.y);
   }
   if ((fg == 6) || (fg == 7))
       printf ("Punto 3 = ");
   else
       printf ("Punto 4 (Inferior derecho) = ");
   printf ("%5d",rec->infder.x);
   printf ("%5d\n",rec->infder.y);
   printf ("Figura tipo = ");
   printf ("%5d\n",rec->figura);
   printf ("Posicion = ");
```

```
printf ("%5d\n",rec->pos);
if (ind == 1) {
    printf ("\nLADOS OBSTACULOS DE LA FIGURA\n\n");
    switch (fg) {
    case 0 : case 1 : case 2 : case 3 : case 4 : case 5 :
        for (i = 0; i < NUMVER; i++)
            if (rec->indob [i] == 1)
                switch (i) {
                case 0 :
                    printf ("Obstaculo en lado p1-p2\n");
                    break;
                case 1 :
                    printf ("Obstaculo en lado p2-p3\n");
                    break;
                case 2 :
                    printf ("Obstaculo en lado p4-p3\n");
                    break;
                case 3 :
                    printf ("Obstaculo en lado p1-p4\n");
                    break;
                }
        break;
    case 6 : case 7 :
        for (i = 0; i < NUMVER - 1; i++)
            if (rec->indob [i] == 1)
                switch (i) {
                case 0 :
                    printf ("Obstaculo en lado p1-p2\n");
                    break;
                case 1 :
                    printf ("Obstaculo en lado p3-p2\n");
                    break;
                case 2 :
                    printf ("Obstaculo en lado p1-p3\n");
                    break;
                }
        break;
    }
}
```

```
extern struct cuadrilatero parte [NUMPAR];
extern int ndn;

/* funcion que efectua la lista ligada de cuadrilateros y triangulos
struct lstcdr *lstfig (ls,ap)
struct lstcdr *ls,*ap;
{
    struct lstcdr *p, *q;
    char *malloc();
    int i,j;

    if (ls == NULL) {
        i = 0 ; ndn++;
        while (parte [i].ident == -1)
            i++;
        parte[i].ident = ndn;
        for (j = i+1; j < NUMPAR; j++)
            if (parte [j].ident != -1)
                parte [j].ident = ++ndn;
        q = (struct lstcdr *) malloc(sizeof (struct lstcdr));
        obelem (&q->nodo,&parte[i],ap,i);
        q->next = NULL;
        ls = q;
        p = ls;
        for (j = i+1; j < NUMPAR; j++)
            if (parte[j].ident != -1) {
                q = (struct lstcdr *) malloc(sizeof (struct lstcdr));
                obelem (&q->nodo,&parte[j],ap,j);
                q->next = NULL;
                p->next = q;
                p = p->next;
            }
    }
    else {
        p = ap;
        i = 0 ;
        while (parte [i].ident == -1)
            i++;
        parte[i].ident = ap->nodo.basico.ident;
        for (j = i+1; j < NUMPAR; j++)
            if (parte [j].ident != -1)
                parte [j].ident = ++ndn;
        for (i = 0 ; i < NUMPAR ; i++)
            if (parte[i].ident != -1) {
                q = (struct lstcdr *) malloc(sizeof (struct lstcdr));
                obelem (&q->nodo,&parte[i],ap,i);
                q->next = p->next;
                p->next = q;
                p = p->next;
            }
    }
    return (ls);


* funcion que obtiene los campos de una estructura elemento */
```

```
/* a partir de los de un cuadrilatero */
obelem (elem,cdro,apr,ipa)
struct elemento *elem;
struct cuadrilatero *cdro;
struct lstcdr *apr;
int ipa;

    int i;

    elem->basico.infizq.x = cdro->infizq.x;
    elem->basico.infizq.y = cdro->infizq.y;
    elem->basico.supizq.x = cdro->supizq.x;
    elem->basico.supizq.y = cdro->supizq.y;
    elem->basico.supder.x = cdro->supder.x;
    elem->basico.supder.y = cdro->supder.y;
    elem->basico.infder.x = cdro->infder.x;
    elem->basico.infder.y = cdro->infder.y;
    for (i = 0; i < NUMPAR; i++)
        elem->basico.indob[i] = cdro->indob[i];
    elem->basico.figura = cdro->figura;
    elem->basico.pos = cdro->pos;
    elem->basico.ident = cdro->ident;
    elem->vecino = formlst (cdro,apr,ipa);


* funcion que forma la estructura de figura para la lista general */
truct lstcua *formlst (cdro,ap,ip)
truct cuadrilatero *cdro;
truct lstcdr *ap;
nt ip;

   struct lstcua *vecvie, *vecact={0}, *vecint(), *vecext();

   vecvie = ap->nodo.vecino;
   if (ap == NULL)
       vecact = vecint (vecact,cdro,ip);
   else {
       vecact = vecext (vecact,cdro,vecvie);
       vecact = vecint (vecact,cdro,ip);
   }
   return (vecact);


* funcion principal que calcula los vecinos de entre las mismas particiones */
truct lstcua *vecint (ls,cua,ind)
truct lstcua *ls;
truct cuadrilatero *cua;
nt ind;

   struct lstcua *formvec();
   struct punto pb1,pb2;
   int i;

   switch (cua->figura) {
   case 0 : case 1 : case 2 : case 3 : case 4 : case 5 :
```

```
        for (i = 0; i < NUMVER; i++)
            if (cua->indob [i] != 1) {
                selcuat (cua,&pb1,&pb2,i);
                ls = formvec (ls,&pb1,&pb2,ind);
            }
        break;
    case 6 : case 7 :
        for (i = 0; i < NUMVER - 1; i++)
            if (cua->indob [i] != 1) {
                seltres (cua,&pb1,&pb2,i);
                ls = formvec (ls,&pb1,&pb2,ind);
            }
        break;
    }
    return (ls);


* funcion auxiliar a vecint que relaciona el lado seleccionado con las *
* particiones restantes */
truct lstcua *formvec (lst,pb1,pb2,ip)
truct lstcua *lst;
truct punto *pb1,*pb2;
nt ip;

    struct lstcua *anade();
    int tb,j;

    obtipo (pb1,pb2,&tb);
    switch (ip) {
    case 0 :
        for (j = 1; j < NUMPAR; j++)
            if (parte [j].ident != -1)
                lst = anade (lst,pb1,pb2,&parte[j],tb);
         break;
    case 1 : {
        if (parte [0].ident != -1)
            lst = anade (lst,pb1,pb2,&parte[0],tb);
        for (j = 2; j < NUMPAR; j++)
            if (parte [j].ident != -1)
                lst = anade (lst,pb1,pb2,&parte[j],tb);
        } break;
    case 2 : {
        for (j = 0; j < 2; j++)
            if (parte [j].ident != -1)
                lst = anade (lst,pb1,pb2,&parte[j],tb);
        for (j = 3; j < NUMPAR; j++)
            if (parte [j].ident != -1)
                lst = anade (lst,pb1,pb2,&parte[j],tb);
        } break;
    case 3 : {
        for (j = 0; j < 3; j++)
            if (parte [j].ident != -1)
                lst = anade (lst,pb1,pb2,&parte[j],tb);
            if (parte [4].ident != -1)
                lst = anade (lst,pb1,pb2,&parte[4],tb);
```

```
        } break;
    case 4 :
        for (j = 0; j < NUMPAR - 1; j++)
            if (parte [j].ident != -1)
                lst = anade (lst,pb1,pb2,&parte[j],tb);
        break;
    }
    return (lst);


* funcion auxiliar a vecint y vecext que anade una figura a la lista */
* de vecinos, si corresponde */
truct lstcua *anade (lsv,pt1,pt2,cua,t)
truct lstcua *lsv;
truct punto *pt1,*pt2;
truct cuadrilatero *cua;
nt t;

    struct punto pa1,pa2;
    int i,ta;

    switch (cua->figura) {
    case 0 : case 1 : case 2 : case 3 : case 4 : case 5 :
        for (i = 0; i < NUMVER; i++) {
            selcuat (cua,&pa1,&pa2,i);
            obtipo (&pa1,&pa2,&ta);
            if ((ta == t) && ((compara (pt1,pt2,&pa1,&pa2,ta)) == 1))
                lsv = lstrec (lsv,cua);
        }
        break;
    case 6 : case 7 :
        for (i = 0; i < NUMVER - 1; i++) {
            seltres (cua,&pa1,&pa2,i);
            obtipo (&pa1,&pa2,&ta);
            if ((ta == t) && ((compara (pt1,pt2,&pa1,&pa2,ta)) == 1))
                lsv = lstrec (lsv,cua);
        }
        break;
    }
    return (lsv);


* funcion que recorre la lista de figuras vecinas de la figura original */
* que se dividio comparando si es vecina de una determinada particion */
truct lstcua *vecext (lst,cdro,lsv)
truct lstcua *lst,*lsv;
truct cuadrilatero *cdro;

    struct lstcua *p;
    struct punto pb1,pb2;
    int tb,i;

    if (lsv == NULL)
        return (NULL);
    else {
```

```
        p = lsv;
        while (p != NULL) {
            switch (cdro->figura) {
            case 0 : case 1 : case 2 : case 3 : case 4 : case 5 :
                for (i = 0; i < NUMVER; i++)
                    if (cdro->indob [i] != 1) {
                        selcuat (cdro,&pb1,&pb2,i);
                        obtipo (&pb1,&pb2,&tb);
                        lst = anade (lst,&pb1,&pb2,&p->conten,tb);
                    }
                break;
            case 6 : case 7 :
                for (i = 0; i < NUMVER - 1; i++)
                    if (cdro->indob [i] != 1) {
                        seltres (cdro,&pb1,&pb2,i);
                        obtipo (&pb1,&pb2,&tb);
                        lst = anade (lst,&pb1,&pb2,&p->conten,tb);
                    }
                break;
            }
            p = p->prox;
        }
    }
    return (lst);


* funcion que actualiza los vecinos de los vecinos de la figura */
* original de entre las particiones de esta */
truct lstcdr *actual (ls,ap)
truct lstcdr *ls,*ap;

    struct lstcua *q;
    struct lstcua *lsv;

    lsv = ap->nodo.vecino;
    if (lsv == NULL)
        return (ls);
    else {
        q = lsv;
        while (q != NULL) {
            ls = busca (ls,ap,q->conten.ident);
            q = q->prox;
        }
    }
    return (ls);


* funcion que encuentra en la lista general la figura dada como dato */
truct lstcdr *busca (lsf,ap,nfig)
truct lstcdr *lsf,*ap;
nt nfig;

    struct lstcdr *p;
    struct lstcua *lsc={0};
```

```
    if (lsf == NULL)
        return (NULL);
    else {
        p = lsf;
        while (p != NULL)
            if (nfig == p->nodo.basico.ident) {
                lsc = p->nodo.vecino;
                lsc = nuevec (ap.lsc,nfig);
                p->nodo.vecino = lsc;
                return (lsf);
            }
            else
                p = p->next;
    }
    return (lsf);


* funcion auxiliar a actual que efectua la tarea de actualizar vecinos */
truct lstcua *nuevec (ap,lsv,cbas)
truct lstcdr *ap;
truct lstcua *lsv;
nt cbas;

    struct lstcua *p,*lv,*ingresa(),*elimina();
    struct lstcdr *q;
    int i;

    if (lsv == NULL)
        return (NULL);
    else {
        p = lsv;
        while (p != NULL)
            if (p->conten.ident == ap->nodo.basico.ident) {
                q = ap;
                for (i = 0; i < NUMPAR; i++)
                    if (parte [i].ident != -1) {
                        q = q->next;
                        lv = q->nodo.vecino;
                        if ((siesta (lv,cbas)) == 1)
                            lsv = ingresa (lsv,p,&parte [i]);
                    }
                lsv = elimina (lsv,p);
                return (lsv);
            }
            else
                p = p->prox;
    }
    return (lsv);


* funcion que elimina un nodo determinado por su apuntador, en la lista */
* general de figuras */
truct lstcdr *borra (ls,ap)
truct lstcdr *ls,*ap;
```

```
    struct lstcdr *p;

    if (ap == ls)
        ls = ap->next;
    else {
        p = ls;
        while (p->next != ap)
            p = p->next;
        p->next = ap->next;
    }
    free ((char *) ap);
    return (ls);


/* funcion que establece si una figura esta en una lista ligada de figuras */
siesta (ls,idcdr)
struct lstcua *ls;
nt idcdr;

    struct lstcua *p;
    int res = 0;

    if (ls == NULL)
        return (res);
    else {
        p = ls;
        while (p != NULL)
            if (p->conten.ident == idcdr) {
                res = 1;
                return (res);
            }
            else
                p = p->prox;
    }
    return (res);


* funcion que ingresa una figura en la lista de figuras vecinas */
truct lstcua *ingresa (lst,ap,rct)
truct lstcua *lst,*ap;
truct cuadrilatero *rct;

    struct lstcua *p,*q;
    char *malloc();

    q = (struct lstcua *) malloc(sizeof (struct lstcua));
    obveci (&q->conten,rct);
    if (lst == ap) {
        q->prox = ap->prox;
        ap->prox = q;
    }
    else {
        p = lst;
        while (p->prox != ap)
            p = p->prox;
```

```
        q->prox = p->prox;
        p->prox = q;
    }
    return (lst);


* funcion que elimina un nodo determinado por su apuntador, en la lista */
* de figuras vecinas */
truct lstcua *elimina (ls,ap)
truct lstcua *ls,*ap;

    struct lstcua *p;

    if (ap == ls)
        ls = ap->prox;
    else {
        p = ls;
        while (p->prox != ap)
            p = p->prox;
        p->prox = ap->prox;
    }
    free ((char *) ap);
    return (ls);
```

```
funcion que establece lá rotacion directa de un cuadrilatero */
:dir (cuadr,cuad,tx,ty.cx.cy)
ruct cuadrilatero *cuad, *cuadr;
t tx, ty, cx, cy;

  int i,po,fg;
  struct punto p , pr;

  po = cuad->pos ; fg = cuad->figura;
  cuadr->pos = 0 ; cuadr->figura = fg;
  switch (po) {
  case 0 :
      obveci (cuadr.cuad);
      break;
  case 1 :
      for (i = 0;i < NUMVER;i++)
          switch (i) {
          case 0 : {
              p.x = cuad->infizq.x ; p.y = cuad->infizq.y;
              rot90 (&p,&pr,cy.tx,ty);
              cuadr->infder.x = pr.x ; cuadr->infder.y = pr.y;
              }
              break;
          case 1 : {
              p.x = cuad->supizq.x ; p.y = cuad->supizq.y;
              rot90 (&p.&pr,cy,tx,ty);
              cuadr->infizq.x = pr.x ; cuadr->infizq.y = pr.y;
              }
              break;
          case 2 :
              if ((fg != 6) && (fg != 7)) {
                  p.x = cuad->supder.x ; p.y = cuad->supder.y;
                  rot90 (&p,&pr,cy,tx,ty);
                  cuadr->supizq.x = pr.x ; cuadr->supizq.y = pr.y;
              }
              break;
          case 3 : {
              p.x = cuad->infder.x ; p.y = cuad->infder.y;
              rot90 (&p,&pr,cy,tx,ty);
              if ((fg == 6) || (fg == 7)) {
                  cuadr->supizq.x = pr.x ; cuadr->supizq.y = pr.y; }
              else {
                  cuadr->supder.x = pr.x ; cuadr->supder.y = pr.y; }
              }
              break;
          }
          break;
  case 2 :
      for (i = 0;i < NUMVER; i++)
          switch (i) {
          case 0: {
              p.x = cuad->infizq.x ; p.y = cuad->infizq.y;
              rot180 (&p,&pr,cx.cy,tx,ty);
              if ((fg == 6) || (fg == 7)) {
                  cuadr->supizq.x = pr.x ; cuadr->supizq.y = pr.y; }
```

```
            else {
                cuadr->supder.x = pr.x ; cuadr->supder.y = pr.y; }
            }
            break;
        case 1 : {
            p.x = cuad->supizq.x ; p.y = cuad->supizq.y;
            rot180 (&p,&pr,cx;cy,tx,ty);
            cuadr->infder.x = pr.x ; cuadr->infder.y = pr.y;
            }
            break;
        case 2 :
            if ((fg != 6) && (fg != 7)) {
                p.x = cuad->supder.x ; p.y = cuad->supder.y;
                rot180 (&p,&pr,cx,cy,tx,ty);
                cuadr->infizq.x = pr.x ; cuadr->infizq.y = pr.y;
            }
            break;
        case 3 : {
            p.x = cuad->infder.x ; p.y = cuad->infder.y;
            rot180 (&p,&pr,cx,cy,tx,ty);
            if ((fg == 6) || (fg == 7)) {
                cuadr->infizq.x = pr.x ; cuadr->infizq.y = pr.y; }
            else {
                cuadr->supizq.x = pr.x ; cuadr->supizq.y = pr.y; }
            }
            break;
        }
        break;
case 3 :
    for (i = 0;i < NUMVER; i++)
        switch (i) {
        case 0 : {
            p.x = cuad->infizq.x ; p.y = cuad->infizq.y;
            rot270 (&p,&pr,cx,tx,ty);
            cuadr->supizq.x = pr.x ; cuadr->supizq.y = pr.y;
            }
            break;
        case 1 : {
            p.x = cuad->supizq.x ; p.y = cuad->supizq.y;
            rot270 (&p,&pr,cx,tx,ty);
            if ((fg == 6) || (fg == 7)) {
                cuadr->infder.x = pr.x ; cuadr->infder.y = pr.y; }
            else {
                cuadr->supder.x = pr.x ; cuadr->supder.y = pr.y; }
            }
            break;
        case 2 :
            if ((fg != 6) && (fg != 7)) {
                p.x = cuad->supder.x ; p.y = cuad->supder.y;
                rot270 (&p,&pr,cx,tx,ty);
                cuadr->infder.x = pr.x ; cuadr->infder.y = pr.y;
            }
            break;
        case 3 : {
            p.x = cuad->infder.x ; p.y = cuad->infder.y;
```

```
                rot270 (&p,&pr.cx,tx,ty);
                cuadr->infizq.x = pr.x ; cuadr->infizq.y = pr.y;
                }
                break;
            }
            break;
    }

* funcion que establece la rotacion directa de un segmrnto */
tdirs (sr,s,po,tx,ty.cx,cy)
truct segmento *s,*sr;
nt po,tx,ty,cx,cy;

    int i,t;
    struct punto p , pr ;

    t = s->tipo;
    switch (po) {
    case 0 : case 2 :
        sr->tipo = s->tipo;
        break;
    case 1 : case 3 :
        if (t == 0)
            sr->tipo = 1;
        else
        if (t == 1)
            sr->tipo = 0;
        else
        if (t == 2)
            sr->tipo = 3;
        else
            sr->tipo = 2;
        break;
    }
    switch (po) {
    case 0 : {
        sr->inicial.x = s->inicial.x ; sr->inicial.y = s->inicial.y;
        sr->final.x = s->final.x ; sr->final.y = s->final.y;
        } break;
    case 1 :
        for (i = 0;i < NUMPT;i++)
            switch (i) {
            case 0 : {
                p.x = s->inicial.x ; p.y = s->inicial.y;
                rot90 (&p,&pr,cy,tx,ty);
                sr->inicial.x = pr.x ; sr->inicial.y = pr.y;
                }
                break;
            case 1 : {
                p.x = s->final.x ; p.y = s->final.y;
                rot90 (&p,&pr,cy,tx,ty);
                sr->final.x = pr.x ; sr->final.y = pr.y;
                }
                break;
```

```
            }
            break;
    case 2 :
        for (i = 0;i < NUMPT; i++)
            switch (i) {
            case 0 : {                              -!
                p.x = s->inicial.x ; p.y = s->inicial.y;
                rot180 (&p,&pr,cx,cy,tx,ty);
                sr->inicial.x = pr.x ; sr->inicial.y = pr.y;
                }
                break;
            case 1 : {
                p.x = s->final.x ; p.y = s->final.y;
                rot180 (&p,&pr,cx,cy,tx,ty);
                sr->final.x = pr.x ; sr->final.y = pr.y;
                }
                break;
            }
            break;
    case 3 :
        for (i = 0;i < NUMPT; i++)
            switch (i) {
            case 0 : {
                p.x = s->inicial.x ; p.y = s->inicial.y;
                rot270 (&p,&pr,cx,tx,ty);
                sr->inicial.x = pr.x ; sr->inicial.y = pr.y;
                }
                break;
            case 1 : {
                p.x = s->final.x ; p.y = s->final.y;
                rot270 (&p,&pr,cx,tx,ty);
                sr->final.x = pr.x ; sr->final.y = pr.y;
                }
                break;
            }
            break;
    }


 funcion que establece la rotacion inversa de un cuadrilatero */
tinv (cuad,cuadr,por,trx,try,crx,cry)
ruct cuadrilatero  *cuadr,*cuad;
t por;
t trx, try, crx, cry;

  int i,fgr,pst;
  struct punto p,pr,p3,p4;

  cuad->pos = por ; fgr = cuad->figura = cuadr->figura;
  switch (por) {
  case 0 :
      obveci (cuad,cuadr);
      break;
  case 1 :
      for (i = 0;i < NUMVER;i++)
```

```
        switch (i) {
        case 0 : {
            pr.x = cuadr->infizq.x ; pr.y = cuadr->infizq.y;
            roti90 (&pr,&p,crx,trx,try);
            cuad->supizq.x = p.x ; cuad->supizq.y = p.y;
            }
            break;
        case 1 : {
            pr.x = cuadr->supizq.x ; pr.y = cuadr->supizq.y;
            roti90 (&pr,&p,crx,trx,try);
            if ((fgr == 6) || (fgr == 7)) {
                cuad->infder.x = p.x ; cuad->infder.y = p.y; }
            else {
                cuad->supder.x = p.x ; cuad->supder.y = p.y; }
            }
            break;
        case 2 :
            if ((fgr != 6) && (fgr != 7)) {
                pr.x = cuadr->supder.x ; pr.y = cuadr->supder.y;
                roti90 (&pr,&p,crx,trx,try);
                cuad->infder.x = p.x ; cuad->infder.y = p.y;
            }
            break;
        case 3 : {
            pr.x = cuadr->infder.x ; pr.y = cuadr->infder.y;
            roti90 (&pr,&p,crx,trx,try);
            cuad->infizq.x = p.x ; cuad->infizq.y = p.y;
            }
            break;
        }
        break;
case 2 :
    for (i = 0;i < NUMVER; i++)
        switch (i) {
        case 0 : {
            pr.x = cuadr->infizq.x ; pr.y = cuadr->infizq.y;
            roti180 (&pr,&p,crx,cry,trx,try);
            if ((fgr == 6) || (fgr == 7)) {
                cuad->infder.x = p.x ; cuad->infder.y = p.y; }
            else {
                cuad->supder.x = p.x ; cuad->supder.y = p.y; }
            }
            break;
        case 1 : {
            pr.x = cuadr->supizq.x ; pr.y = cuadr->supizq.y;
            roti180 (&pr,&p,crx,cry,trx,try);
            if ((fgr == 6) || (fgr == 7)) {
                cuad->infizq.x = p.x ; cuad->infizq.y = p.y; }
            else {
                cuad->infder.x = p.x ; cuad->infder.y = p.y; }
            }
            break;
        case 2 :
            if ((fgr != 6) && (fgr != 7)) {
                pr.x = cuadr->supder.x ; pr.y = cuadr->supder.y;
```

```
                    roti180 (&pr,&p,crx,cry,trx,try);
                    cuad->infizq.x = p.x ; cuad->infizq.y = p.y;
                }
                break;
            case 3 : {
                pr.x = cuadr->infder.x ; pr.y = cuadr->infder.y;
                roti180 (&pr,&p,crx,cry,trx,try);
                cuad->supizq.x = p.x ; cuad->supizq.y = p.y;
                }
                break;
            }
            break;
    case 3 :
        for (i = 0;i < NUMVER; i++)
            switch (i) {
            case 0 : {
                pr.x = cuadr->infizq.x ; pr.y = cuadr->infizq.y;
                roti270 (&pr,&p,cry,trx,try);
                cuad->infder.x = p.x ; cuad->infder.y = p.y;
                }
                break;
            case 1 : {
                pr.x = cuadr->supizq.x ; pr.y = cuadr->supizq.y;
                roti270 (&pr,&p,cry,trx,try);
                cuad->infizq.x = p.x ; cuad->infizq.y = p.y;
                }
                break;
            case 2 :
                if ((fgr != 6) && (fgr != 7)) {
                    pr.x = cuadr->supder.x ; pr.y = cuadr->supder.y;
                    roti270 (&pr,&p,cry,trx,try);
                    cuad->supizq.x = p.x ; cuad->supizq.y = p.y;
                }
                break;
            case 3 : {
                pr.x = cuadr->infder.x ; pr.y = cuadr->infder.y;
                roti270 (&pr,&p,cry,trx,try);
                if ((fgr == 6) || (fgr == 7)) {
                    cuad->supizq.x = p.x ; cuad->supizq.y = p.y; }
                else {
                    cuad->supder.x = p.x ; cuad->supder.y = p.y; }
                }
                break;
            }
            break;
    }
    descol (cuad,&p,&pr,&p3,&p4,&fgr,&pst);
    figpos (&p,&pr,&p3,&p4,&fgr,&pst);
    cuad->pos = pst;


* funcion que establece la rotacion inversa de un segmrnto */
tinvs (s,sr,por,trx,try,crx,cry)
truct segmento  *sr,*s;
nt por,trx,try,crx,cry;
```

```
int i,tr;
struct punto p, pr;

tr = sr->tipo;
switch (por) {
case 0 : case 2 :
    s->tipo = sr->tipo;
    break;
case 1 : case 3 :
    if (tr == 0)
        s->tipo = 1;
    else
    if (tr == 1)
        s->tipo = 0;
    else
    if (tr == 2)
        s->tipo = 3;
    else
        s->tipo = 2;
    break;
}
switch (por) {
case 0 : {
    s->inicial.x = sr->inicial.x ; s->inicial.y = sr->inicial.y;
    s->final.x = sr->final.x ; s->final.y = sr->final.y;
    } break;
case 1 :
    for (i = 0;i < NUMPT;i++)
        switch (i) {
        case 0 : {
            pr.x = sr->inicial.x ; pr.y = sr->inicial.y;
            roti90 (&pr,&p,crx,trx,try);
            s->inicial.x = p.x ; s->inicial.y = p.y;
            }
            break;
        case 1 : {
            pr.x = sr->final.x ; pr.y = sr->final.y;
            roti90 (&pr,&p,crx,trx,try);
            s->final.x = p.x ; s->final.y = p.y;
            }
            break;
        }
        break;
case 2 :
    for (i = 0;i < NUMPT; i++)
        switch (i) {
        case 0 : {
            pr.x = sr->inicial.x ; pr.y = sr->inicial.y;
            roti180 (&pr,&p,crx,cry,trx,try);
            s->inicial.x = p.x ; s->inicial.y = p.y;
            }
            break;
        case 1 : {
            pr.x = sr->final.x ; pr.y = sr->final.y;
```

```
                roti180 (&pr,&p,crx,cry,trx,try);
                s->final.x = p.x ; s->final.y = p.y;
                }
                break;
            }
            break;
    case 3 :
        for (i = 0;i < NUMPT; i++)
            switch (i) {
            case 0 : {
                pr.x = sr->inicial.x ; pr.y = sr->inicial.y;
                roti270 (&pr,&p,cry,trx,try);
                s->inicial.x = p.x ; s->inicial.y = p.y;
                }
                break;
            case 1 : {
                pr.x = sr->final.x ; pr.y = sr->final.y;
                roti270 (&pr,&p,cry,trx,try);
                s->final.x = p.x ; s->final.y = p.y;
                }
                break;
            }
            break;
    }

* funcion que obtiene las constantes para la traslacion */
nstrans (cuad,dx,dy)
ruct cuadrilatero *cuad;
t *dx, *dy;

   struct punto p1, p2;

   p1.x = cuad->infizq.x ; p1.y = cuad->infizq.y;
   p2.x = cuad->supizq.x ; p2.y = cuad->supizq.y;
   if (p1.x <= p2.x)
       *dx = p1.x;
   else
       *dx = p2.x;
   p2.x = cuad->infder.x ; p2.y = cuad->infder.y;
   if (p1.y <= p2.y)
       *dy = p1.y;
   else
       *dy = p2.y;


* funcion que obtiene las constantes para la rotacion */
nsrot (cuad,dx,dy)
truct cuadrilatero *cuad;
t *dx, *dy;

   struct punto p1, p2;
   int fg;

   fg = cuad->figura;
```

```
    switch (cuad->pos) {
    case 0 : case 1 : {
        p1.x = cuad->infizq.x ; p1.y = cuad->infizq.y;
        p2.x = cuad->infder.x ;
        if (fg == 4)
            p2.x = cuad->supder.x;
        if (fg == 5)
            p1.x = cuad->supizq.x;
        *dx = p2.x - p1.x;
        if (fg == 4) {
            p1.y = cuad->infder.y;
            p2.y = cuad->supizq.y; }
        else
        if (fg == 5)
            p2.y = cuad->supder.y;
        else
            p2.y = cuad->supizq.y;
        *dy = p2.y - p1.y;
        } break;
    case 2 : {
        p1.x = cuad->supizq.x ; p1.y = cuad->supizq.y;
        if ((fg == 6) || (fg == 7))
            p2.x = cuad->infder.x ;
        else
            p2.x = cuad->supder.x ;
        *dx = p2.x - p1.x;
        p2.y = cuad->infizq.y;
        *dy = p1.y - p2.y;
        } break;
    case 3 : {
        p1.x = cuad->infizq.x ;
        p2.x = cuad->infder.x ; p2.y = cuad->infder.y;
        *dx = p2.x - p1.x;
        if ((fg == 6) || (fg == 7))
            p1.y = cuad->supizq.y;
        else
            p1.y = cuad->supder.y;
        *dy = p1.y - p2.y;
        }
        break;
    }


* funcion que establece la rotacion directa a 90 grados */
ot90 (v,vr,c1,c2,c3)
truct punto *v, *vr;
nt c1,c2,c3;

   vr->x = - v->y + c1 + c2 + c3;
   vr->y = v->x + c3 - c2;


* funcion que establece la rotacion directa a 180 grados */
ot180 (v,vr,c1,c2,c3,c4)
truct punto *v, *vr;
```

```
.nt c1,c2,c3,c4;

   vr->x = - v->x + c1 + 2*c3;
   vr->y = - v->y + c2 + 2*c4;


* funcion que establece la rotacion directa a 270 grados */
'ot270 (v,vr,c1,c2,c3)
;truct punto *v, *vr;
.nt c1,c2,c3;

   vr->x = v->y + c2 - c3;
   vr->y = - v->x + c1 + c2 + c3;


* funcion que establece la rotacion inversa a 90 grados */
oti90 (vr,v,c1,c2,c3)
truct punto *vr, *v;
nt c1,c2,c3;

   v->x = vr->y - c3 + c2;
   v->y = c1 + c2 + c3 - vr->x;


* funcion que establece la rotacion inversa a 180 grados */
oti180 (vr,v,c1,c2,c3,c4)
truct punto *vr, *v;
nt c1,c2,c3,c4;

   v->x = c1 + 2*c3 - vr->x;
   v->y = c2 + 2*c4 - vr->y;


* funcion que establece la rotacion inversa a 270 grados */
oti270 (vr,v,c1,c2,c3)
truct punto *vr, *v;
nt c1,c2,c3;

   v->x = c1 + c2 + c3 - vr->y;
   v->y = vr->x + c3 - c2;
```

```
* funcion que determina cual es la region geometrica a dividirse */
truct lstcdr *donde (ls,s,ind1,ind2)
truct lstcdr *ls;
truct segmento *s;
nt *ind1,*ind2;

   struct lstcdr *p;
   struct punto p1,p2,p3,p4,pi,pf;
   int fig,po,t,ahi,haydos;

   desco2 (s,&pi,&pf,&t);
   if (ls == NULL) {
       *ind1 = 0;
       *ind2 = 0;
       return (NULL);
   }
   else {
       p = ls;
       while (p != NULL) {
           desco1 (&p->nodo.basico,&p1,&p2,&p3,&p4,&fig,&po);
           switch (fig) {
           case 0 :
               ahi = enfig0 (&p1,&p2,&p3,&p4,&pi,&pf,t,&haydos);
               break;
           case 1 :
           case 6 :
               ahi = enfig16 (&p1,&p2,&p3,&p4,&pi,&pf,t,po,&haydos);
               break;
           case 2 :
               ahi = enfig2 (&p1,&p2,&p3,&p4,&pi,&pf,t,po,&haydos);
               break;
           case 3 :
           case 7 :
               ahi = enfig37 (&p1,&p2,&p3,&p4,&pi,&pf,t,po,&haydos);
               break;
           case 4 :
               ahi = enfig4 (&p1,&p2,&p3,&p4,&pi,&pf,t,po,&haydos);
               break;
           case 5 :
               ahi = enfig5 (&p1,&p2,&p3,&p4,&pi,&pf,t,po,&haydos);
               break;
           }
           if ((ahi == 1) || (ahi == -1)) {
               *ind1 = ahi;
               *ind2 = haydos;
               return (p);
           } else
               p = p->next;
       }
       printf("peligro. el punto no esta en las figuras.\n");
       for (;;) ;
   }


funcion que considera que el punto inicial del obstaculo cae en un */
```

```
* rectangulo */
nfig0 (p1,p2,p3,p4,pa,pb,t,mas)
truct punto *p1,*p2,*p3,*p4,*pa,*pb;
nt t,*mas;

   int res = 0;

   *mas = 0;
   switch (t) {
   case 0 :
       if ((pa->x >= p1->x) && (pa->x < p3->x) &&
           (pa->y >= p1->y) && (pa->y <= p3->y))
           res = 1;
       if ((res == 1) && ((pa->y == p1->y) || (pa->y == p3->y)))
           *mas = 1;
       break;
   case 1 :
       if ((pa->x >= p1->x) && (pa->x <= p3->x) &&
           (pa->y >= p1->y) && (pa->y < p3->y))
           res = 1;
       if ((res == 1) && ((pa->x == p1->x) || (pa->x == p3->x)))
           *mas = 1;
       break;
   case 2 :
       if ((pa->x >= p1->x) && (pa->x < p3->x) &&
           (pa->y >= p1->y) && (pa->y < p3->y))
           res = 1;
       break;
   case 3 :
       if ((pa->x > p1->x) && (pa->x <= p3->x) &&
           (pa->y >= p1->y) && (pa->y < p3->y))
           res = 1;
       break;
   }
   if ((res == 1) && ((pb->x > p4->x) || (pb->y > p3->y) || (pb->x < p1->x)))
       res = -1;
   return (res);


* funcion que considera que el punto inicial del obstaculo cae en un */
* trapecio o triangulo rectangulo izquierdo */
nfig16 (p1,p2,p3,p4,pa,pb,t,po,mas)
truct punto *p1,*p2,*p3,*p4,*pa,*pb;
nt po,t,*mas;

   int di,df,res = 0;

   *mas = 0;
   switch (t) {
   case 0 : {
       switch (po) {
       case 0 : {
           di = pa->y + pa->x - p4->y - p4->x;
           df = pb->y + pb->x - p4->y - p4->x;
           if ((pa->x >= p1->x) && (di < 0) &&
```

```
            (pa->y >= p1->y) && (pa->y <= p2->y))
            res = 1;
        if ((res == 1) && ((pa->y == p1->y) || (pa->y == p2->y)))
            *mas = 1;
        } break;
    case 1 : {
        di = pa->y - pa->x + p1->x - p1->y;
        df = pb->y - pb->x + p1->x - p1->y;
        if ((pa->x >= p1->x) && (pa->x < p4->x) && (di > 0) &&
            (pa->y <= p2->y) && (pa->y > p1->y))
            res = 1;
        if ((res == 1) && (pa->y == p2->y))
            *mas = 1;
        } break;
    case 2 : {
        di = pa->y + pa->x - p1->y - p1->x;
        if ((di >= 0) && (pa->x < p4->x) &&
            (pa->y >= p1->y) && (pa->y <= p2->y))
            res = 1;
        if ((res == 1) && ((pa->y == p1->y) || (pa->y == p2->y)))
            *mas = 1;
        } break;
    case 3 : {
        di = pa->y - pa->x + p2->x - p2->y;
        if ((pa->x >= p1->x) && (di <= 0) &&
            (pa->x < p4->x) && (pa->y >= p1->y))
            res = 1;
        if ((res == 1) && (pa->y == p1->y))
            *mas = 1;
        } break;
    }
    if ((res == 1) && ((pb->x > p4->x) || ((df > 0) && (po == 0)) ||
        ((po == 1) && (df < 0))))
        res = -1;
    } break;
case 1 : {
    switch (po) {
    case 0 : {
        di = pa->y + pa->x - p4->y - p4->x;
        df = pb->y + pb->x - p4->y - p4->x;
        if ((pa->x >= p1->x) && (pa->x < p4->x) &&
            (pa->y >= p1->y) && (pa->y < p2->y) && (di < 0))
            res = 1;
        if ((res == 1) && (pa->x == p1->x))
            *mas = 1;
        } break;
    case 1 : {
        di = pa->y - pa->x + p1->x - p1->y;
        if ((di >= 0) && (pa->y < p2->y) &&
            (pa->x >= p1->x) && (pa->x <= p4->x))
            res = 1;
        if ((res == 1) && ((pa->x == p1->x) || (pa->x == p4->x)))
            *mas = 1;
        } break;
    case 2 : {
```

```
        di = pa->y + pa->x - p1->y - p1->x;
        if ((di >= 0) && (pa->y >= p1->y) &&
            (pa->y < p2->y) && (pa->x <= p4->x))
            res = 1;
        if ((res == 1) && (pa->x == p4->x))
            *mas = 1;
        } break;
    case 3 : {
        di = pa->y - pa->x + p2->x - p2->y;
        df = pb->y - pb->x + p2->x - p2->y;
        if ((pa->x >= p1->x) && (pa->x <= p4->x) &&
            (pa->y >= p1->y) && (di < 0))
            res = 1;
        if ((res == 1) && ((pa->x == p1->x) || (pa->x == p4->x)))
            *mas = 1;
        } break;
    }
    if ((res == 1) && ((pb->y > p2->y) || (df > 0)))
        res = -1;
    } break;
case 2 : {
    switch (po) {
    case 0 : {
        di = pa->y + pa->x - p4->y - p4->x;
        df = pb->y + pb->x - p4->y - p4->x;
        if ((pa->x >= p1->x) && (di < 0) &&
            (pa->y >= p1->y) && (pa->y < p2->y))
            res = 1;
        } break;
    case 1 : {
        di = pa->y - pa->x + p1->x - p1->y;
        if ((pa->x >= p1->x) && (di >= 0) &&
            (pa->y < p2->y) && (pa->x < p4->x))
            res = 1;
        if ((res == 1) && (di == 0))
            *mas = 1;
        } break;
    case 2 : {
        di = pa->y + pa->x - p1->y - p1->x;
        if ((di >= 0) && (pa->y >= p1->y) &&
            (pa->y < p2->y) && (pa->x < p4->x))
            res = 1;
        } break;
    case 3 : {
        di = pa->y - pa->x + p2->x - p2->y;
        if ((pa->x >= p1->x) && (pa->x < p4->x) &&
            (di <= 0) && (pa->y >= p1->y))
            res = 1;
        if ((res == 1) && (di == 0))
            *mas = 1;
        } break;
    }
    if ((res == 1) && ((pb->x > p4->x) || (pb->y > p2->y) || (df > 0)))
        res = -1;
    } break;
```

```
  case 3 : {
      switch (po) {
      case 0 : {
          di = pa->y + pa->x - p4->y - p4->x;
          if ((pa->x > p1->x) && (pa->y >= p1->y) &&
              (pa->y < p2->y) && (di <= 0))
              res = 1;
          if ((res == 1) && (di == 0))
              *mas = 1;
          } break;
      case 1 : {
          di = pa->y - pa->x + p1->x - p1->y;
          if ((di >= 0) && (pa->x <= p4->x) &&
              (pa->x > p1->x) && (pa->y < p2->y))
              res = 1;
          } break;
      case 2 : {
          di = pa->y + pa->x - p1->y - p1->x;
          if ((di >= 0) && (pa->y >= p1->y) &&
              (pa->x <= p4->x) && (pa->y < p2->y))
              res = 1;
          if ((res == 1) && (di == 0))
              *mas = 1;
          } break;
      case 3 : {
          di = pa->y - pa->x + p2->x - p2->y;
          df = pb->y - pb->x + p2->x - p2->y;
          if ((pa->x <= p4->x) && (pa->x > p1->x) &&
              (pa->y >= p1->y) && (di < 0))
              res = 1;
          } break;
      }
      if ((res == 1) && ((pb->x < p1->x) || (pb->y > p2->y) || (df > 0)))
          res = -1;
      } break;
  }
  return (res);


' funcion que considera que el punto inicial del obstaculo cae en un */
' trapecio rectangulo derecho */
ıfig2 (p1,p2,p3,p4,pa,pb,t,po,mas)
truct punto *p1,*p2,*p3,*p4,*pa,*pb;
ıt po.t,*mas;

  int di,df,res = 0;

  *mas = 0;
  switch (t) {
  case 0 : {
      switch (po) {
      case 0 : {
          di = pa->y - pa->x + p1->x - p1->y;
          if ((di <= 0) && (pa->x < p3->x) &&
              (pa->y >= p1->y) && (pa->y <= p3->y))
```

```
            res = 1;
        if ((res == 1) && ((pa->y == p1->y) || (pa->y == p3->y)))
            *mas = 1;
        } break;
    case 1 : {
        di = pa->y + pa->x - p3->x - p3->y;
        df = pb->y + pb->x - p3->x - p3->y;
        if ((pa->x >= p1->x) && (pa->x < p3->x) &&
            (di < 0) && (pa->y >= p1->y))
            res = 1;
        if ((res == 1) && (pa->y == p1->y))
            *mas = 1;
        } break;
    case 2 : {
        di = pa->y - pa->x + p4->x - p4->y;
        df = pb->y - pb->x + p4->x - p4->y;
        if ((pa->x >= p1->x) && (di > 0) &&
            (pa->y >= p1->y) && (pa->y <= p3->y))
            res = 1;
        if ((res == 1) && ((pa->y == p1->y) || (pa->y == p3->y)))
            *mas = 1;
        } break;
    case 3 : {
        di = pa->y + pa->x - p4->x - p4->y;
        if ((pa->x >= p1->x) && (di >= 0) &&
            (pa->x < p4->x) && (pa->y <= p3->y))
            res = 1;
        if ((res == 1) && (pa->y == p3->y))
            *mas = 1;
        } break;
    }
    if ((res == 1) && ((pb->x > p4->x) || ((po == 1) && (df > 0)) ||
        ((po == 2) && (df < 0))))
        res = -1;
    } break;
case 1 : {
    switch (po) {
    case 0 : {
        di = pa->y - pa->x + p1->x - p1->y;
        df = pb->y - pb->x + p1->x - p1->y;
        if ((pa->x > p1->x) && (pa->x <= p3->x) &&
            (pa->y >= p1->y) && (pa->y < p3->y) && (di < 0))
            res = 1;
        if ((res == 1) && (pa->x == p3->x))
            *mas = 1;
        } break;
    case 1 : {
        di = pa->y + pa->x - p3->x - p3->y;
        df = pb->y + pb->x - p3->x - p3->y;
        if ((pa->x >= p1->x) && (pa->x <= p4->x) &&
            (pa->y >= p1->y) && (di < 0))
            res = 1;
        if ((res == 1) && ((pa->x == p1->x) || (pa->x == p3->x)))
            *mas = 1;
        } break;
```

```
    case 2 : {
        di = pa->y - pa->x + p4->x - p4->y;
        if ((pa->x >= p1->x) && (pa->y >= p1->y) &&
            (di >= 0) && (pa->y < p3->y))
            res = 1;
        if ((res == 1) && (pa->x == p1->x))
            *mas = 1;
        } break;
    case 3 : {
        di = pa->y + pa->x - p4->x - p4->y;
        if ((di >= 0) && (pa->y < p3->y) &&
            (pa->x >= p1->x) && (pa->x <= p4->x))
            res = 1;
        if ((res == 1) && ((pa->x == p1->x) || (pa->x == p3->x)))
            *mas = 1;
        } break;
    }
    if ((res == 1) && ((pb->y > p3->y) || (df > 0)))
        res = -1;
    } break;
case 2 : {
    switch (po) {
    case 0 : {
        di = pa->y - pa->x + p1->x - p1->y;
        if ((di <= 0) && (pa->x < p3->x) &&
            (pa->y >= p1->y) && (pa->y < p3->y))
            res = 1;
        if ((res == 1) && (di == 0))
            *mas = 1;
        } break;
    case 1 : {
        di = pa->y + pa->x - p3->x - p3->y;
        df = pb->y + pb->x - p3->x - p3->y;
        if ((pa->x >= p1->x) && (pa->x < p4->x) &&
            (di < 0) && (pa->y >= p1->y))
            res = 1;
        } break;
    case 2 : {
        di = pa->y - pa->x + p4->x - p4->y;
        if ((pa->x >= p1->x) && (pa->y >= p1->y) &&
            (pa->y < p3->y) && (di >= 0))
            res = 1;
        if ((res == 1) && (di == 0))
            *mas = 1;
        } break;
    case 3 : {
        di = pa->y + pa->x - p4->x - p4->y;
        if ((pa->x >= p1->x) && (pa->x < p4->x) &&
            (di >= 0) && (pa->y < p3->y))
            res = 1;
        } break;
    }
    if ((res == 1) && ((pb->x > p4->x) || (pb->y > p3->y) || (df > 0)))
        res = -1;
    } break;
```

```
   case 3 : {
       switch (po) {
       case 0 : {
           di = pa->y - pa->x + p1->x - p1->y;
           df = pb->y - pb->x + p1->x - p1->y;
           if ((pa->x <= p3->x) && (pa->y >= p1->y) &&
               (pa->y < p3->y) && (di < 0))
               res = 1;
           } break;
       case 1 : {
           di = pa->y + pa->x - p3->x - p3->y;
           if ((pa->x <= p3->x) && (pa->x > p1->x) &&
               (pa->y >= p1->y) && (di <= 0))
               res = 1;
           if ((res == 1) && (di == 0))
               *mas = 1;
           } break;
       case 2 : {
           di = pa->y - pa->x + p4->x - p4->y;
           if ((pa->y >= p1->y) && (pa->x > p1->x) &&
               (di >= 0) && (pa->y < p3->y))
               res = 1;
           } break;
       case 3 : {
           di = pa->y + pa->x - p4->x - p4->y;
           if ((pa->x <= p3->x) && (pa->x > p1->x) &&
               (di >= 0) && (pa->y < p3->y))
               res = 1;
           if ((res == 1) && (di == 0))
               *mas = 1;
           } break;
       }
       if ((res == 1) && ((pb->x < p1->x) || (pb->y > p3->y) || (df > 0)))
           res = -1;
       } break;
   }
   return (res);


* funcion que considera que el punto inicial del obstaculo cae en un */
* trapecio o triangulo rectangulo isosceles */
nfig37 (p1,p2,p3,p4,pa,pb,t,po,mas)
truct punto *p1,*p2,*p3,*p4,*pa,*pb;
nt po,t,*mas;

   int di,di1,df,df1,res = 0;

   *mas = 0;
   switch (t) {
   case 0 : {
       switch (po) {
       case 0 : {
           di = pa->y + pa->x - p4->y - p4->x;
           di1 = pa->y - pa->x + p1->x - p1->y;
           df = pb->y + pb->x - p4->y - p4->x;
```

```
        if ((di1 <= 0) && (di < 0) &&
            (pa->y >= p1->y) && (pa->y <= p2->y))
            res = 1;
        if ((res == 1) && ((pa->y == p1->y) || (pa->y == p2->y)))
            *mas = 1;
        } break;
    case 1 : {
        di = pa->y - pa->x + p1->x - p1->y;
        df = pb->y - pb->x + p1->x - p1->y;
        di1 = pa->y + pa->x - p3->x - p3->y;
        df1 = pb->y + pb->x - p3->x - p3->y;
        if ((pa->x >= p1->x) && (pa->x < p4->x) &&
            (di > 0) && (di1 < 0))
            res = 1;
        } break;
    case 2 : {
        di = pa->y + pa->x - p1->y - p1->x;
        di1 = pa->y - pa->x + p4->x - p4->y;
        df1 = pb->y - pb->x + p4->x - p4->y;
        if ((di >= 0) && (di1 > 0) &&
            (pa->y >= p1->y) && (pa->y <= p2->y))
            res = 1;
        if ((res == 1) && ((pa->y == p1->y) || (pa->y == p2->y)))
            *mas = 1;
        } break;
    case 3 : {
        di = pa->y - pa->x + p2->x - p2->y;
        di1 = pa->y + pa->x - p4->x - p4->y;
        if ((pa->x >= p1->x) && (pa->x < p4->x) &&
            (di <= 0) && (di1 >= 0))
            res = 1;
        } break;
    }
    if ((res == 1) && (((po == 2) && (df1 < 0)) || ((df > 0) && (po == 0)) |
        ((po == 1) && ((df < 0) || (df1 > 0) || (pb->x > p4->x))) ||
        ((po == 3) && (pb->x > p4->x))))
        res = -1;
    } break;
case 1 : {
    switch (po) {
    case 0 : {
        di = pa->y + pa->x - p4->y - p4->x;
        df = pb->y + pb->x - p4->y - p4->x;
        di1 = pa->y - pa->x + p1->x - p1->y;
        df1 = pb->y - pb->x + p1->x - p1->y;
        if ((pa->y >= p1->y) && (pa->y < p2->y) &&
            (di1 < 0) && (di < 0))
            res = 1;
        } break;
    case 1 : {
        di = pa->y - pa->x + p1->x - p1->y;
        di1 = pa->y + pa->x - p3->x - p3->y;
        df1 = pb->y + pb->x - p3->x - p3->y;
        if ((di >= 0) && (di1 < 0) &&
            (pa->x >= p1->x) && (pa->x <= p4->x))
```

```
            res = 1;
        if ((res == 1) && ((pa->x == p1->x) || (pa->x == p4->x)))
            *mas = 1;
        } break;
    case 2 : {
        di = pa->y + pa->x - p1->y - p1->x;
        di1 = pa->y - pa->x + p4->x - p4->y;
        if ((di >= 0) && (pa->y >= p1->y) && (di1 >= 0) &&
            (pa->y < p2->y))
            res = 1;
        } break;
    case 3 : {
        di = pa->y - pa->x + p2->x - p2->y;
        df = pb->y - pb->x + p2->x - p2->y;
        di1 = pa->y + pa->x - p4->x - p4->y;
        if ((pa->x >= p1->x) && (pa->x <= p4->x) &&
            (di1 >= 0) && (di < 0))
            res = 1;
        if ((res == 1) && ((pa->x == p1->x) || (pa->x == p4->x)))
            *mas = 1;
        } break;
    }
    if ((res == 1) && ((pb->y > p2->y) || (df1 > 0) || (df > 0)))
        res = -1;
    } break;
case 2 : {
    switch (pq) {
    case 0 : {
        di = pa->y + pa->x - p4->y - p4->x;
        df = pb->y + pb->x - p4->y - p4->x;
        di1 = pa->y - pa->x + p1->x - p1->y;
        if ((di1 <= 0) && (di < 0) &&
            (pa->y >= p1->y) && (pa->y < p2->y))
            res = 1;
        if ((res == 1) && (di1 == 0))
            *mas = 1;
        } break;
    case 1 : {
        di = pa->y - pa->x + p1->x - p1->y;
        di1 = pa->y + pa->x - p3->x - p3->y;
        df1 = pb->y + pb->x - p3->x - p3->y;
        if ((pa->x >= p1->x) && (di >= 0) &&
            (di1 < 0) && (pa->x < p4->x))
            res = 1;
        if ((res == 1) && (di == 0))
            *mas = 1;
        } break;
    case 2 : {
        di = pa->y + pa->x - p1->y - p1->x;
        di1 = pa->y - pa->x + p4->x - p4->y;
        if ((di >= 0) && (pa->y >= p1->y) &&
            (di1 >= 0) && (pa->y < p2->y))
            res = 1;
        if ((res == 1) && (di1 == 0))
            *mas = 1;
```

```
        } break;
    case 3 : {
        di = pa->y - pa->x + p2->x - p2->y;
        di1 = pa->y + pa->x - p4->x - p4->y;
        if ((pa->x >= p1->x) && (pa->x < p4->x) &&
            (di <= 0) && (di1->= 0))
            res = 1;
        if ((res == 1) && (di == 0))
            *mas = 1;
        } break;
    }
    if ((res == 1) && ((pb->x > p4->x) || (pb->y > p2->y) ||
        (df > 0) || (df1 > 0)))
        res = -1;
    } break;
case 3 : {
    switch (po) {
    case 0 : {
        di = pa->y + pa->x - p4->y - p4->x;
        di1 = pa->y - pa->x + p1->x - p1->y;
        df1 = pb->y - pb->x + p1->x - p1->y;
        if ((di1 < 0) && (pa->y >= p1->y) &&
            (pa->y < p2->y) && (di <= 0))
            res = 1;
        if ((res == 1) && (di == 0))
            *mas = 1;
        } break;
    case 1 : {
        di = pa->y - pa->x + p1->x - p1->y;
        di1 = pa->y + pa->x - p3->x - p3->y;
        if ((di >= 0) && (pa->x <= p4->x) &&
            (pa->x > p1->x) && (di1 <= 0))
            res = 1;
        if ((res == 1) && (di1 == 0))
            *mas = 1;
        } break;
    case 2 : {
        di = pa->y + pa->x - p1->y - p1->x;
        di1 = pa->y - pa->x + p4->x - p4->y;
        if ((di >= 0) && (pa->y >= p1->y) &&
            (di1 >= 0) && (pa->y < p2->y))
            res = 1;
        if ((res == 1) && (di == 0))
            *mas = 1;
        } break;
    case 3 : {
        di = pa->y - pa->x + p2->x - p2->y;
        df = pb->y - pb->x + p2->x - p2->y;
        di1 = pa->y + pa->x - p4->x - p4->y;
        if ((pa->x <= p4->x) && (pa->x > p1->x) &&
            (di1 >= 0) && (di < 0))
            res = 1;
        if ((res == 1) && (di1 == 0))
            *mas = 1;
        } break;
```

```
        }
        if ((res == 1) && ((pb->y > p2->y) || (df1 > 0) ||
            (pb->x < p1->x) || (df > 0)))
            res = -1;
        } break;
    }
    return (res);


* funcion que considera que el punto inicial del obstaculo cae en un */
* romboide agudo */
nfig4 (p1,p2,p3,p4,pa,pb,t,po,mas)
truct punto *p1,*p2,*p3,*p4,*pa,*pb;
nt po,t,*mas;

    int di,di1,df,df1,res = 0;

    *mas = 0;
    switch (t) {
    case 0 : {
        switch (po) {
        case 0 : {
            di = pa->y - pa->x - p4->y + p4->x;
            df = pb->y - pb->x + p4->x - p4->y;
            di1 = pa->y - pa->x + p1->x - p1->y;
            if ((di1 <= 0) && (di > 0) &&
                (pa->y >= p1->y) && (pa->y <= p2->y))
                res = 1;
            if ((res == 1) && ((pa->y == p1->y) || (pa->y == p2->y)))
                *mas = 1;
            } break;
        case 1 : {
            di = pa->y + pa->x - p3->y - p3->x;
            df = pb->y + pb->x - p3->x - p3->y;
            di1 = pa->y + pa->x - p4->x - p4->y;
            if ((pa->x >= p1->x) && (di1 >= 0) &&
                (pa->x < p4->x) && (di < 0))
                res = 1;
            } break;
        }
        if ((res == 1) && (((po == 0) && (df < 0)) ||
            ((po == 1) && ((df > 0) || (pb->x > p4->x)))))
            res = -1;
        } break;
    case 1 : {
        switch (po) {
        case 0 : {
            di = pa->y - pa->x - p4->y + p4->x;
            di1 = pa->y - pa->x + p1->x - p1->y;
            df1 = pb->y - pb->x + p1->x - p1->y;
            if (((di >= 0) || (pa->y >= p1->y)) &&
                (di1 < 0) && (pa->y < p2->y))
                res = 1;
            } break;
        case 1 : {
```

```
        di = pa->y + pa->x - p3->y - p3->x;
        df = pb->y + pb->x - p3->x - p3->y;
        di1 = pa->y + pa->x - p4->x - p4->y;
        if ((pa->x >= p1->x) && (pa->x <= p4->x) &&
            (di1 >= 0) && (di < 0))
            res = 1;
        if ((res == 1) && ((pa->x == p1->x) || (pa->x == p4->x)))
            *mas = 1;
        } break;
    }
    if ((res == 1) && ((df1 > 0) || (pb->y > p2->y) || (df > 0)))
        res = -1;
    } break;
case 2 : {
    switch (po) {
    case 0 : {
        di = pa->y - pa->x - p4->y + p4->x;
        di1 = pa->y - pa->x + p1->x - p1->y;
        if ((di1 <= 0) && (di >= 0) &&
            (pa->y >= p1->y) && (pa->y < p2->y))
            res = 1;
        if ((res == 1) && ((di == 0) || (di1 == 0)))
            *mas = 1;
        } break;
    case 1 : {
        di = pa->y + pa->x - p3->y - p3->x;
        df = pb->y + pb->x - p3->x - p3->y;
        di1 = pa->y + pa->x - p4->x - p4->y;
        if ((pa->x >= p1->x) && (pa->x < p4->x) &&
            (di1 >= 0) && (di < 0))
            res = 1;
        } break;
    }
    if ((res == 1) && ((pb->y > p2->y) || (df > 0) || (pb->x > p4->x)))
        res = -1;
    } break;
case 3 : {
    switch (po) {
    case 0 : {
        di = pa->y - pa->x - p4->y + p4->x;
        di1 = pa->y - pa->x + p1->x - p1->y;
        df1 = pb->y - pb->x + p1->x - p1->y;
        if ((di >= 0) && (pa->y >= p1->y) &&
            (di1 < 0) && (pa->y < p2->y))
            res = 1;
        } break;
    case 1 : {
        di = pa->y + pa->x - p3->y - p3->x;
        di1 = pa->y + pa->x - p4->x - p4->y;
        if ((pa->x <= p4->x) && (pa->x > p1->x) &&
            (di1 >= 0) && (di <= 0))
            res = 1;
        if ((res == 1) && ((di == 0) || (di1 == 0)))
            *mas = 1;
        } break;
```

```
        }
        if ((res == 1) && (((po == 0) && ((df1 > 0) || (pb->y > p2->y)))
            ((po == 1) && (pb->x < p1->x))))
            res = -1;
        } break;
    }
    return (res);


* funcion que considera que el punto inicial del obstaculo cae en un */
* romboide obtuso */
nfigS (p1,p2,p3,p4,pa,pb,t,po,mas)
truct punto *p1,*p2,*p3,*p4,*pa,*pb;
nt po,t,*mas;

    int di,di1,df,df1,res = 0;

    *mas = 0;
    switch (t) {
    case 0 : {
        switch (po) {
        case 0 : {
            di = pa->y + pa->x - p4->y - p4->x;
            df = pb->y + pb->x - p4->y - p4->x;
            di1 = pa->y + pa->x - p1->x - p1->y;
            if ((di1 >= 0) && (di < 0) &&
                (pa->y >= p1->y) && (pa->y <= p2->y))
                res = 1;
            if ((res == 1) && ((pa->y == p1->y) || (pa->y == p2->y)))
                *mas = 1;
            } break;
        case 1 : {
            di = pa->y - pa->x - p1->y + p1->x;
            df = pb->y - pb->x - p1->y + p1->x;
            di1 = pa->y - pa->x + p2->x - p2->y;
            if ((pa->x >= p1->x) && (pa->x < p4->x) &&
                (di1 <= 0) && (di > 0))
                res = 1;
            } break;
        }
        if ((res == 1) && (((po == 0) && (df > 0)) ||
            ((po == 1) && ((pb->x > p4->x) || (df < 0)))))
            res = -1;
        } break;
    case 1 : {
        switch (po) {
        case 0 : {
            di = pa->y + pa->x - p4->y - p4->x;
            df = pb->y + pb->x - p4->y - p4->x;
            di1 = pa->y + pa->x - p1->x - p1->y;
            if (((di1 >= 0) || (pa->y >= p1->y)) &&
                (di < 0) && (pa->y < p2->y))
                res = 1;
            } break;
        case 1 : {
```

```
        di = pa->y - pa->x - p1->y + p1->x;
        di1 = pa->y - pa->x + p2->x - p2->y;
        df1 = pb->y - pb->x + p2->x - p2->y;
        if ((pa->x >= p1->x) && (pa->x <= p4->x) &&
            (di >= 0) && (di1 < 0))
            res = 1;
        if ((res == 1) && ((pa->x == p1->x) || (pa->x == p4->x)))
            *mas = 1;
        } break;
    }
    if ((res == 1) && ((pb->y > p3->y) || (df > 0) || (df1 > 0)))
        res = -1;
    } break;
case 2 : {
    switch (po) {
    case 0 : {
        di = pa->y + pa->x - p4->y - p4->x;
        df = pb->y + pb->x - p4->y - p4->x;
        di1 = pa->y + pa->x - p1->x - p1->y;
        if ((di1 >= 0) && (di < 0) &&
            (pa->y >= p1->y) && (pa->y < p2->y))
            res = 1;
        } break;
    case 1 : {
        di = pa->y - pa->x - p1->y + p1->x;
        di1 = pa->y - pa->x + p2->x - p2->y;
        if ((pa->x >= p1->x) && (pa->x < p4->x) &&
            (di >= 0) && (di1 <= 0))
            res = 1;
        if ((res == 1) && ((di == 0) || (di1 == 0)))
            *mas = 1;
        } break;
    }
    if ((res == 1) && ((pb->y > p3->y) || (df > 0) || (pb->x > p4->x)))
        res = -1;
    } break;
case 3 : {
    switch (po) {
    case 0 : {
        di = pa->y + pa->x - p4->y - p4->x;
        di1 = pa->y + pa->x - p1->x - p1->y;
        if ((di1 >= 0) && (pa->y >= p1->y) &&
            ((di <= 0) && (pa->y < p2->y)))
            res = 1;
        if ((res == 1) && ((di == 0) || (di1 == 0)))
            *mas = 1;
        } break;
    case 1 : {
        di = pa->y - pa->x - p1->y + p1->x;
        di1 = pa->y - pa->x + p2->x - p2->y;
        df1 = pb->y - pb->x + p2->x - p2->y;
        if ((pa->x <= p4->x) && (pa->x > p1->x) &&
            (di >= 0) && (di1 < 0))
            res = 1;
        } break;
```

```
    }
    if ((res == 1) && ((pb->y > p3->y) || (pb->x < p1->x) || (df1 > 0)))
        res = -1;
    } break;
}
return (res);
```

```
* funcion que recorre la lista de vecinos de la figura dividida */
* verificando si el obstaculo continua en alguno de ellos */
truct lstcdr *continua (ls,s,lsv,ind)
truct lstcdr *ls;
truct segmento *s;
truct lstcua *lsv;
nt *ind;

   struct lstcdr *ap = NULL;
   struct lstcua *q;
   int ahi = 1;

   *ind = 1;
   if (lsv == NULL) {
       *ind = 0;
       return (NULL);
   }
   else {
       q = lsv;
       while ((q != NULL) && (ahi == 1)) {
           ahi = atravi (&q->conten,s);
           if ((ahi == 0) || (ahi == -1)) {
               ap = ubica (ls,q->conten.ident);
               *ind = ahi;
               return (ap);
           }
           else
               q = q->prox;
       }
   }
   return (ap);


* funcion que establece si una figura puede dividirse, si el */
* obstaculo considerado atraviesa mas de una region */
travi (cuad,seg)
truct cuadrilatero *cuad;
truct segmento *seg;

   struct punto p1,p2,p3,p4,pi,pf;
   struct segmento sr;
   struct cuadrilatero cuar ;
   int fig,po,por,ti,tx,ty,cx,cy,ind;

   po = cuad->pos;
   constrans (cuad,&tx,&ty);
   consrot (cuad,&cx,&cy);
   rotdir (&cuar,cuad,tx,ty,cx,cy);
   desco1 (&cuar.&p1,&p2,&p3,&p4,&fig,&por);
   rtdirs (&sr,seg,po,tx,ty,cx,cy);
   desco2 (&sr,&pi,&pf,&ti);
   switch (fig) {
   case 0 :
       ind = sigue0 (&p1,&p2,&p3,&p4.&pi,&pf,ti);
       break;
```

```
   case 1 : case 6 :
       ind = sigue16 (&p1,&p2,&p3,&p4,&pi,&pf,ti);
       break;
   case 2 :
       ind = sigue2 (&p1,&p2,&p3,&p4,&pi,&pf,ti);
       break;
   case 3 : case 7 :
       ind = sigue37 (&p1,&p2,&p3,&p4,&pi,&pf,ti);
       break;
   case 4 :
       ind = sigue4 (&p1,&p2,&p3,&p4,&pi,&pf,ti);
       break;
   case 5 :
       ind = sigue5 (&p1,&p2,&p3,&p4,&pi,&pf,ti);
       break;
   }
   return (ind);


* funcion que considera que un obstaculo continua en un rectangulo */
igue0 (p1,p2,p3,p4,pa,pb,t)
truct punto *p1,*p2,*p3,*p4,*pa,*pb;
nt t;

   int yi,yd,res = 1;

   switch (t) {
   case 0 :
       if ((pa->y >= p1->y) && (pa->y <= p3->y))
           if ((pa->x < p1->x) && (pb->x > p1->x) && (pb->x <= p3->x))
               res = 0;
           else
           if ((pa->x < p1->x) && (pb->x > p3->x))
               res = -1;
       break;
   case 1 :
       if ((pa->x >= p1->x) && (pa->x <= p3->x))
           if ((pa->y < p1->y) && (pb->y > p1->y) && (pb->y <= p3->y))
               res = 0;
           else
           if ((pa->y < p1->y) && (pb->y > p3->y))
               res = -1;
       break;
   case 2 : {
       yi = p1->x + pa->y - pa->x ; yd = p3->x + pa->y - pa->x;
       if ((yi >= p1->y) && (yi < p3->y))
           if (((pa->x < p1->x) && (pb->x > p1->x) && (pb->x <= p3->x))
               ((pa->x < p1->x) && (pb->x > p1->x) && (pb->y <= p3->y)))
               res = 0;
           else
           if (((pa->x < p1->x) && (pb->x > p3->x)) ||
               ((pa->x < p1->x) && (pb->y > p3->y)))
               res = -1;
           else
               res = 1;
```

```
        else
        if ((yi < p1->y) && (yd > p1->y))
            if (((pa->y < p1->y) && (pb->y > p1->y) && (pb->y <= p3->y)) ||
                ((pa->y < p1->y) && (pb->y > p1->y) && (pb->x <= p3->x)))
                res = 0;
            else
            if (((pa->y < p1->y) && (pb->y > p3->y)) ||
                ((pa->y < p1->y) && (pb->x > p3->x)))
                res = -1;
        } break;
    case 3 : {
        yi = pa->y + pa->x - p1->x ; yd = pa->y + pa->x - p3->x;
        if ((yd >= p1->y) && (yd < p3->y))
            if (((pa->x > p3->x) && (pb->x < p3->x) && (pb->x >= p1->x)) ||
                ((pa->x > p3->x) && (pb->x < p3->x) && (pb->y <= p3->y)))
                res = 0;
            else
            if (((pa->x > p3->x) && (pb->x < p1->x)) ||
                ((pa->x > p3->x) && (pb->y > p3->y)))
                res = -1;
            else
                res = 1;
        else
        if ((yi > p1->y) && (yd < p1->y))
            if (((pa->y < p1->y) && (pb->y > p1->y) && (pb->y <= p3->y)) ||
                ((pa->y < p1->y) && (pb->y > p1->y) && (pb->x >= p1->x)))
                res = 0;
            else
            if (((pa->y < p1->y) && (pb->y > p3->y)) ||
                ((pa->y < p1->y) && (pb->x < p1->x)))
                res = -1;
        } break;
    }
    return (res);


* funcion que considera que un obstaculo continua en un trapecio o */
* triangulo rectangulo izquierdo */
igue16 (p1,p2,p3,p4,pa,pb,t)
truct punto *p1,*p2,*p3,*p4,*pa,*pb;
nt t;

    int di,df,xi,res = 1;

    switch (t) {
    case 0 : {
        di = pa->y + pa->x - p4->y - p4->x;
        df = pb->y + pb->x - p4->y - p4->x;
        if ((pa->y >= p1->y) && (pa->y <= p2->y))
            if (pa->x < pb->x)
                if ((pa->x < p1->x) && (pb->x > p1->x) && (df <= 0))
                    res = 0;
                else
                if ((pa->x < p1->x) && (df > 0))
                    res = -1;
```

```
            else
                res = 1;
        else
        if (pb->x < pa->x)
            if ((di > 0) && (df < 0) && (pb->x >= p1->x))
                res = 0;
            else
            if ((di > 0) && (pb->x < p1->x))
                res = -1;
    } break;
case 1 : {
    di = pa->y + pa->x - p4->y - p4->x;
    df = pb->y + pb->x - p4->y - p4->x;
    if ((pa->x >= p1->x) & (pa->x < p4->x))
        if (pa->y < pb->y)
            if ((pa->y < p1->y) && (pb->y > p1->y) &&
               (pb->y <= p2->y) && (df <= 0))
                res = 0;
            else
            if ((pa->y < p1->y) && ((pb->y > p2->y) || (df > 0)))
                res = -1;
            else
                res = 1;
        else
        if (pb->y < pa->y)
            if (((pa->y > p2->y) || (di > 0)) && (pb->y < p2->y) &&
                (df < 0) && (pb->y >= p1->y))
                res = 0;
            else
            if (((pa->y > p2->y) || (di > 0)) && (pb->y < p1->y))
                res = -1;
    } break;
case 2 :
    if (pa->y < pb->y)
        res = nor1645 (p1,p2,p3,p4,pa,pb);
    else
        res = inv1645 (p1,p2,p3,p4,pa,pb);
    break;
case 3 : {
    xi = pa->y + pa->x - p1->y;
    di = pa->y + pa->x - p4->y - p4->x;
    if ((xi > p1->x) && (xi <= p4->x) && (di <= 0))
        if (pa->y < pb->y)
            if (((pa->y < p1->y) && (pb->y > p1->y) && (pb->y <= p2->y)) ||
                ((pa->y < p1->y) && (pb->y > p1->y) && (pb->x >= p1->x)))
                res = 0;
            else
            if (((pa->y < p1->y) && (pb->y > p2->y)) ||
                ((pa->y < p1->y) && (pb->x < p1->x)))
                res = -1;
            else
                res = 1;
        else
        if (pb->y < pa->y)
            if (((pa->y > p2->y) && (pb->y < p2->y) &&
```

```
                    (pb->x > p1->x) && (pb->y >= p1->y)) ||
                    ((pa->x < p1->x) && (pb->x > p1->x) &&
                    (pb->y < p2->y) && (pb->y >= p1->y)))
                    res = 0;
                else
                if (((pa->y > p2->y) && (pb->y < p1->y)) ||
                    ((pa->x < p1->x) && (pb->y < p1->y)))
                    res = -1;
        } break;
    }
    return (res);


' funcion auxiliar a sigue16 que considera a un obstaculo oblicuo */
' a 45 grados en su posicion normal */
or1645 (p1,p2,p3,p4,pa,pb)
truct punto *p1,*p2,*p3,*p4,*pa,*pb;

    int yi,yd,df,con = 1;

    df = pb->y + pb->x - p4->y - p4->x;
    yi = p1->x + pa->y - pa->x ; yd = p4->x + pa->y - pa->x;
    if ((yi >= p1->y) && (yi < p2->y))
        if (((pa->x < p1->x) && (pb->x > p1->x) && (df <= 0)) ||
            ((pa->x < p1->x) && (pb->x > p1->x) && (pb->y <= p2->y)))
            con = 0;
        else
        if (((pa->x < p1->x) && (df > 0)) ||
            ((pa->x < p1->x) && (pb->y > p2->y)))
            con = -1;
        else
            con = 1;
        else
        if ((yi < p1->y) && (yd > p1->y))
            if (((pa->y < p1->y) && (pb->y > p1->y) && (df <= 0)) ||
                ((pa->y < p1->y) && (pb->y > p1->y) && (pb->y <= p2->y)))
                con = 0;
            else
            if (((pa->y < p1->y) && (df > 0)) ||
                ((pa->y < p1->y) && (pb->y > p2->y)))
                con = -1;
    return (con);


' funcion auxiliar a sigue16 que considera a un obstaculo oblicuo */
' a 45 grados en su posicion invertida */
nv1645 (p1,p2,p3,p4,pa,pb)
truct punto *p1,*p2,*p3,*p4,*pa,*pb;

    int yi,xi,di,df,con = 1;

    xi = p1->y + pa->x - pa->y ; yi = p1->x + pa->y - pa->x;
    df = pb->y + pb->x - p4->y - p4->x;
    di = pa->y + pa->x - p4->y - p4->x;
    if ((xi >= p1->x) && (xi < p4->x))
```

```
        if (((di > 0) || (pa->y > p2->y)) && (pb->y < p2->y) &&
            (df < 0) && (pb->y >= p1->y))
            con = 0;
        else
        if (((di > 0) || (pa->y > p2->y)) && (pb->y < p1->y))
            con = -1;
        else
            con = 1;
    else
    if ((yi >= p1->y) && (yi < p2->y))
        if (((di > 0) || (pa->y > p2->y)) && (pb->y < p2->y) &&
            (df < 0) && (pb->x >= p1->x))
            con = 0;
        else
        if (((di > 0) || (pa->y > p2->y)) && (pb->x < p1->x))
            con = -1;
    return (con);


* funcion que considera que el obstaculo continua en un trapecio */
* rectangulo derecho */
igue2 (p1,p2,p3,p4,pa,pb,t)
truct punto *p1,*p2,*p3,*p4,*pa,*pb;
nt t;

    int xi,di,df,res = 1;

    switch (t) {
    case 0 : {
       di = pa->y - pa->x + p1->x - p1->y;
       df = pb->y - pb->x + p1->x - p1->y;
       if ((pa->y >= p1->y) && (pa->y <= p3->y))
           if (pa->x < pb->x)
               if ((di > 0) && (df < 0) && (pb->x <= p3->x))
                   res = 0;
               else
               if ((di > 0) && (pb->x > p3->x))
                   res = -1;
               else
                   res = 1;
           else
           if (pb->x < pa->x)
               if ((pa->x > p3->x) && (pb->x < p3->x) && (df <= 0))
                   res = 0;
               else
               if ((pa->x > p3->x) && (df > 0))
                   res = -1;
       } break;
    case 1 : {
       di = pa->y - pa->x + p1->x - p1->y;
       df = pb->y - pb->x + p1->x - p1->y;
       if ((pa->x > p1->x) && (pa->x <= p3->x))
           if (pa->y < pb->y)
               if ((pa->y < p1->y) && (pb->y > p1->y) &&
                   (pb->y <= p3->y) && (df <= 0))
```

```
                    res = 0;
                else
                if ((pa->y < p1->y) && ((pb->y > p3->y) || (df > 0)))
                    res = -1;
                else
                    res = 1;
            else
            if (pb->y < pa->y)
                if (((pa->y > p2->y) || (di > 0)) && (pb->y < p2->y) &&
                    (df < 0) && (pb->y >= p1->y))
                    res = 0;
                else
                if (((pa->y > p2->y) || (di > 0)) && (pb->y < p1->y))
                    res = -1;
        } break;
    case 2 : {
        xi = p1->y + pa->x - pa->y;
        di = pa->y - pa->x + p1->x - p1->y;
        if ((xi >= p1->x) && (xi < p3->x) && (di <= 0))
            if (pa->y < pb->y)
                if ((pa->y < p1->y) && (pb->y > p1->y) &&
                    (pb->x <= p3->x) && (pb->y <= p3->y))
                    res = 0;
                else
                if ((pa->y < p1->y) && (pb->y > p3->y))
                    res = -1;
                else
                    res = 1;
            else
            if (pb->y < pa->y)
                if (((pa->y > p2->y) && (pb->y < p2->y) &&
                    (pb->x < p3->x) && (pb->y >= p1->y)) ||
                    ((pa->x > p3->x) && (pb->x < p3->x) &&
                    (pb->y < p2->y) && (pb->y >= p1->y)))
                    res = 0;
                else
                if (((pa->y > p2->y) && (pb->y < p1->y)) ||
                    ((pa->x > p3->x) && (pb->y < p1->y)))
                    res = -1;
        } break;
    case 3 :
        if (pa->y < pb->y)
            res = nor2135 (p1,p2,p3,p4,pa,pb);
        else
            res = inv2135 (p1,p2,p3,p4,pa,pb);
        break;
    }
    return (res);


* funcion auxiliar a sigue2 que considera a un obstaculo oblicuo */
* a 135 grados en su posicion normal */
or2135 (p1,p2,p3,p4,pa,pb)
truct punto *p1,*p2,*p3,*p4,*pa,*pb;
```

```
    int yi,yd,df,con = 1;

    yi = pa->y + pa->x - p1->x ; yd = pa->y + pa->x - p3->x;
    df = pb->y - pb->x + p1->x - p1->y;
    if ((yd >= p1->y) && (yd < p3->y))
        if ((pa->x > p3->x) && (pb->x < p3->x) &&
            (pb->y <= p3->y) && (df <= 0))
            con = 0;
        else
        if ((pa->x > p3->x) && ((df > 0) || (pb->y > p3->y)))
            con = -1;
        else
            con = 1;
    else
    if ((yd < p1->y) && (yi > p1->y))
        if ((pa->y < p1->y) && (pb->y > p1->y) &&
            (pb->y <= p3->y) && (df <= 0))
            con = 0;
        else
        if ((pa->y < p1->y) && ((df > 0) || (pb->y > p3->y)))
            con = -1;
    return (con);
}

/* funcion auxiliar a sigue2 que considera a un obstaculo oblicuo */
/* a 135 grados en su posicion invertida */
inv2135 (p1,p2,p3,p4,pa,pb)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
{
    int xi,yd,di,df,con = 1;

    xi = pa->y + pa->x - p1->y ; yd = pa->y + pa->x - p3->x;
    di = pa->y - pa->x + p1->x - p1->y;
    df = pb->y - pb->x + p1->x - p1->y;
    if ((xi > p1->x) && (xi <= p3->x))
        if (((di > 0) || (pa->y > p2->y)) && (pb->y < p2->y) &&
            (df < 0) && (pb->y >= p1->y))
            con = 0;
        else
        if (((di > 0) || (pa->y > p2->y)) && (pb->y < p1->y))
            con = -1;
        else
            con = 1;
    else
    if ((yd >= p1->y) && (yd < p3->y))
        if (((di > 0) || (pa->y > p2->y)) && (pb->y < p2->y) &&
            (df < 0) && (pb->x <= p3->x))
            con = 0;
        else
        if (((di > 0) || (pa->y > p2->y)) && (pb->x > p3->x))
            con = -1;
    return (con);
}

/* funcion que considera que un obstaculo continua en un trapecio */
```

```
 * o triangulo rectangulo isosceles */
figue37 (p1,p2,p3,p4,pa,pb,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int t;

    int di,di1,df,df1,xi,res = 1;

    di = pa->y + pa->x - p4->y - p4->x;
    df = pb->y + pb->x - p4->y - p4->x;
    di1 = pa->y - pa->x + p1->x - p1->y;
    df1 = pb->y - pb->x + p1->x - p1->y;
    switch (t) {
    case 0 :
        if ((pa->y >= p1->y) && (pa->y <= p2->y))
            if (pa->x < pb->x)
                if ((di1 > 0) && (df <= 0) && (df1 < 0))
                    res = 0;
                else
                if ((di1 > 0) && (df1 > 0))
                    res = -1;
                else
                    res = 1;
            else
            if (pb->x < pa->x)
                if ((di > 0) && (df < 0) && (df1 <= 0))
                    res = 0;
                else
                if ((di > 0) && (df1 > 0))
                    res = -1;
        break;
    case 1 :
        if ((pa->x > p1->x) && (pa->x < p4->x))
            if (pa->y < pb->y)
                if ((pa->y < p1->y) && (pb->y > p1->y) &&
                    (pb->y <= p2->y) && (df1 <= 0) && (df <= 0))
                    res = 0;
                else
                if ((pa->y < p1->y) && ((pb->y > p2->y) ||
                    (df1 > 0) && (df > 0)))
                    res = -1;
                else
                    res = 1;
            else
            if (pb->y < pa->y)
                if (((di1 > 0) || (pa->y > p2->y) || (di > 0)) && (df1 < 0) &&
                    (pb->y < p2->y) && (df < 0) && (pb->y >= p1->y))
                    res = 0;
                else
                if (((di1 > 0) || (pa->y > p2->y) || (di > 0)) &&
                    (pb->y < p1->y))
                    res = -1;
        break;
    case 2 : {
        xi = p1->y + pa->x - pa->y;
        if ((xi >= p1->x) && (xi < p4->x))
```

```
            if (pa->y < pb->y)
                if ((pa->y < p1->y) && (pb->y > p1->y) &&
                    (pb->y <= p2->y) && (df <= 0))
                    res = 0;
                else
                if ((pa->y < p1->y) && ((pb->y > p2->y) || (df > 0)))
                    res = -1;
                else
                    res = 1;
            else
            if (pb->y < pa->y)
                if (((pa->y > p2->y) || (di > 0)) && (pb->y < p2->y) &&
                    (df < 0) && (pb->y >= p1->y))
                    res = 0;
                else
                if (((pa->y > p2->y) || (di > 0)) && (pb->y < p1->y))
                    res = -1;
        } break;
    case 3 : {
        xi = pa->x + pa->y - p1->y;
        if ((xi > p1->x) && (xi <= p4->x))
            if (pa->y < pb->y)
                if ((pa->y < p1->y) && (pb->y > p1->y) &&
                    (pb->y <= p2->y) && (df1 <= 0))
                    res = 0;
                else
                if ((pa->y < p1->y) && ((pb->y > p2->y) || (df1 > 0)))
                    res = -1;
                else
                    res = 1;
            else
            if (pb->y < pa->y)
                if (((pa->y > p2->y) || (di1 > 0)) && (pb->y < p2->y) &&
                    (df1 < 0) && (pb->y >= p1->y))
                    res = 0;
                else
                if (((pa->y > p2->y) || (di1 > 0)) && (pb->y < p1->y))
                    res = -1;
        } break;
    }
    return (res);
}

/* funcion que considera que el obstaculo continua en un romboide */
/* agudo */
sigue4 (p1,p2,p3,p4,pa,pb,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int t;
{
    int di,di1,df,df1,xi,res = 1;

    di = pa->y - pa->x - p4->y + p4->x;
    df = pb->y - pb->x + p4->x - p4->y;
    di1 = pa->y - pa->x + p1->x - p1->y;
    df1 = pb->y - pb->x + p1->x - p1->y;
```

```
switch (t) {
case 0 :
    if ((pa->y >= p1->y) && (pa->y <= p2->y))
        if (pa->x < pb->x)
            if ((di1 > 0) && (df1 < 0) && (df >= 0))
                res = 0;
            else
            if ((di1 > 0) && (df < 0))
                res = -1;
            else
                res = 1;
        else
        if (pb->x < pa->x)
            if ((di < 0) && (df > 0) && (df1 <= 0))
                res = 0;
            else
            if ((di < 0) && (df1 > 0))
                res = -1;
    break;
case 1 :
    if ((pa->x > p1->x) && (pa->x < p3->x))
        if (((di < 0) || (pa->y < p1->y)) && (pb->y > p1->y) &&
            (df > 0) && (pb->y <= p3->y) && (df1 <= 0))
            res = 0;
        else
        if (((di < 0) || (pa->y < p1->y)) &&
            ((pb->y > p3->y) || (df1 > 0)))
            res = -1;
    break;
case 2 : {
    xi = p1->y + pa->x - pa->y;
    if ((xi >= p1->x) && (xi <= p4->x))
        if (pa->y < pb->y)
            if ((pa->y < p1->y) && (pb->y > p1->y) && (pb->y <= p3->y))
                res = 0;
            else
            if ((pa->y < p1->y) && (pb->y > p3->y))
                res = -1;
            else
                res = 1;
        else
        if (pb->y < pa->y)
            if ((pa->y > p3->y) && (pb->y < p3->y) && (pb->y >= p1->y))
                res = 0;
            else
            if ((pa->y > p3->y) && (pb->y < p1->y))
                res = -1;
    } break;
case 3 : {
    xi = pa->x + pa->y - p1->y;
    if ((xi > p1->x) && (xi < 2*p3->x - p4->x))
        if (((pa->y < p1->y) || (di < 0)) && (pb->y > p1->y) &&
            (df > 0) && (pb->y <= p3->y) && (df1 <= 0))
            res = 0;
        else
```

```
            if (((pa->y < p1->y) || (di < 0)) &&
                ((pb->y > p3->y) || (df1 > 0)))
                res = -1;
        } break;
    }
    return (res);
}

/* funcion que considera que el obstaculo continua en un romboide */
/* obtuso */
sigue5 (p1,p2,p3,p4,pa,pb,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int t;
{
    int di,di1,df,df1,xi,res = 1;

    di = pa->y + pa->x - p4->y - p4->x;
    df = pb->y + pb->x - p4->y - p4->x;
    di1 = pa->y + pa->x - p1->x - p1->y;
    df1 = pb->y + pb->x - p1->x - p1->y;
    switch (t) {
    case 0 :
        if ((pa->y >= p1->y) && (pa->y <= p2->y))
            if (pa->x < pb->x)
                if ((di1 < 0) && (df1 > 0) && (df <= 0))
                    res = 0;
                else
                if ((di1 < 0) && (df > 0))
                    res = -1;
                else
                    res = 1;
            else
            if (pb->x < pa->x)
                if ((di > 0) && (df < 0) && (df1 >= 0))
                    res = 0;
                else
                if ((di > 0) && (df1 < 0))
                    res = -1;
        break;
    case 1 :
        if ((pa->x > p2->x) && (pa->x < p4->x))
            if (((di1 < 0) || (pa->y < p1->y)) && (df1 > 0) &&
                (pb->y > p1->y) && (pb->y <= p3->y) && (df <= 0))
                res = 0;
            else
            if (((di1 < 0) || (pa->y < p1->y)) &&
                ((pb->y > p3->y) || (df > 0)))
                res = -1;
        break;
    case 2 : {
        xi = p1->y + pa->x - pa->y;
        if ((xi > 2*p2->x - p1->x) && (xi < p4->x))
            if (pa->y < pb->y)
                if (((pa->y < p1->y) || (di1 < 0)) && (pb->y > p1->y) &&
                    (df1 > 0) && (pb->y <= p3->y) && (df <= 0))
```

```
                res = 0;
            else
            if (((pa->y < p1->y) || (di1 < 0)) &&
                ((pb->y > p3->y) || (df > 0)))
                res = -1;
            else
                res = 1;
        else
        if (pb->y < pa->y)
            if (((pa->y > p3->y) || (di > 0)) && (pb->y < p3->y) &&
                (df < 0) && (pb->y >= p1->y) && (df1 >= 0))
                res = 0;
            else
            if (((pa->y > p3->y) || (di > 0)) &&
                ((pb->y < p1->y) || (df1 < 0)))
                res = -1;
    } break;
case 3 : {
    xi = pa->x + pa->y - p1->y;
    if ((xi >= p1->x) && (xi <= p4->x))
        if ((pa->y < p1->y) && (pb->y > p1->y) && (pb->y <= p3->y))
            res = 0;
        else
        if ((pa->y < p1->y) && (pb->y > p3->y))
            res = -1;
    } break;
}
return (res);
}
```

```
/* funcion que considera la presencia de obstaculos horizontales *
obshor (cuad,seg)
struct cuadrilatero *cuad;
struct segmento *seg;
{
    struct cuadrilatero cuar;                    -f
    struct segmento segr;
    struct punto p1,p2,p3,p4,pi,pf;
    int fig,por,po,t,tr,tx,ty,cx,cy;

    po = cuad->pos ; t = seg->tipo;
    switch (po) {
    case 0 : {
        desco1 (cuad,&p1,&p2,&p3,&p4,&fig,&po);
        desco2 (seg,&pi,&pf,&t);
        switch (fig) {
        case 0 :
            rechor (&p1,&p2,&p3,&p4,&pi,&pf,fig,t);
            break;
        case 1 : case 6 :
            if (pi.x < pf.x)
                t1t6hor (&p1,&p2,&p3,&p4,&pi,&pf,fig,t);
            else
                t1t6hor (&p1,&p2,&p3,&p4,&pf,&pi,fig,t);
            break;
        case 2 :
            if (pi.x < pf.x)
                tra2hor (&p1,&p2,&p3,&p4,&pi,&pf,fig,t);
            else
                tra2hor (&p1,&p2,&p3,&p4,&pf,&pi,fig,t);
            break;
        case 3 : case 7 :
            if (pi.x < pf.x)
                t3t7hor (&p1,&p2,&p3,&p4,&pi,&pf,fig,t);
            else
                t3t7hor (&p1,&p2,&p3,&p4,&pf,&pi,fig,t);
            break;
        case 4 :
            if (pi.x < pf.x)
                rom4hor (&p1,&p2,&p3,&p4,&pi,&pf,fig,t);
            else
                rom4hor (&p1,&p2,&p3,&p4,&pf,&pi,fig,t);
            break;
        case 5 :
            if (pi.x < pf.x)
                rom5hor (&p1,&p2,&p3,&p4,&pi,&pf,fig,t);
            else
                rom5hor (&p1,&p2,&p3,&p4,&pf,&pi,fig,t);
            break;
        }
    }
    break;
    case 1 : case 2 : case 3 : {
        constrans (cuad,&tx,&ty);
        consrot (cuad,&cx,&cy);
```

```
        rotdir (&cuar,cuad.tx,ty,cx,cy);
        rtdirs (&segr,seg,po,tx,ty,cx,cy);
        desco1 (&cuar,&p1,&p2,&p3,&p4,&fig,&por);
        desco2 (&segr,&pi,&pf,&tr);
        switch (tr) {
        case 0 :
            switch (fig) {
            case 0 :
                break;
            case 1 : case 6 :
                t1t6hor (&p1,&p2,&p3,&p4,&pf,&pi,fig,t);
                break;
            case 2 :
                tra2hor (&p1,&p2,&p3,&p4,&pf,&pi,fig,t);
                break;
            case 3 : case 7 :
                t3t7hor (&p1,&p2,&p3,&p4,&pf,&pi,fig,t);
                break;
            case 4 :
                rom4hor (&p1,&p2,&p3,&p4,&pf,&pi,fig,t);
                break;
            case 5 :
                rom5hor (&p1,&p2,&p3,&p4,&pf,&pi,fig,t);
                break;
            }
        break;
        case 1 :
            obsver (&cuar,&segr);
            break;
        case 2 : case 3 :
            break;
        }
        constrans (&cuar,&tx,&ty) ; consrot (&cuar,&cx,&cy);
        invierte (po,tx,ty,cx,cy);
    }
    break;
    }
}

/* funcion que divide un rectangulo por un obstaculo horizontal */
rechor (p1,p2,p3,p4,pa,pb,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int fig,t;
{
    if ((pa->x <= p1->x) && (pb->x >= p3->x))
        dive20 (p1,p2,p3,p4,p1->x,pa->y,p3->x,pa->y,t);
    else
    if ((pa->x <= p1->x) && (pb->x < p3->x))
        dive30 (p1,p2,p3,p4,pb,p1->x,pa->y,pb->x,p3->y,p1->y,fig,t);
    else
    if ((pa->x > p1->x) && (pb->x >= p3->x))
        dive31 (p1,p2,p3,p4,pa,pa->x,p3->y,p3->x,pb->y,p1->y,fig,t);
    else
    if ((pa->x > p1->x) && (pb->x < p3->x))
        dive40 (p1,p2,p3,p4,pa,pb,p3->y,p3->y,p1->y,p1->y,fig,t);
```

```
}

/* funcion que divide un trapecio rectangulo izquierdo o un triangulo */
/* rectangulo izquierdo por un obstaculo horizontal */
t1t6hor (p1,p2,p3,p4,pa,pb,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int fig,t;
{
    int ds,c1,c2,c3;

    ds = pb->x + pb->y - p4->x - p4->y;
    c1 = p4->x + p4->y - pa->x ; c2 = p4->x + p4->y - pb->x;
    c3 = p4->x + p4->y - pa->y;
    if ((pa->x <= p1->x) && (ds >= 0))
        aebe16h (p1,p2,p3,p4,pa,pb,fig,t);
    else
    if ((pa->x <= p1->x) && (ds < 0))
        aizbinh (p1,p2,p3,p4,pa,pb,c2,fig,t);
    else
    if ((pa->x > p1->x) && (ds >= 0))
        ainbdeh (p1,p2,p3,p4,pa,pb,c1,c3,fig,t);
    else
    if ((pa->x > p1->x) && (ds < 0))
        ainbinh (p1,p2,p3,p4,pa,pb,c1,c2,fig,t);
}

/* funcion auxiliar a t1t6nor que supone a los extremos del obstaculo */
/* exteriores a la figura */
aebe16h (p1,p2,p3,p4,pa,pb,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int fig,t;
{
    int c;

    c = p4->x + p4->y - pb->y;
    if (fig == 1)
        dive20 (p1,p2,p3,p4,p1->x,pa->y,c,pa->y,t);
    else
    if (fig == 6)
        dive21 (p1,p2,p3,p4,p1->x,pa->y,c,pa->y,t);
}

/* funcion auxiliar a t1t6hor que considera al extremo inicial del obstaculo */
/* a la izquierda y al extremo final interior al cuadrilatero */
aizbinh (p1,p2,p3,p4,pa,pb,c,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int c,fig,t;
{
        if ((fig == 1) && (pb->x <= p3->x))
            dive30 (p1,p2,p3,p4,pb,p1->x,pa->y,pb->x,p3->y,p1->y,fig,t);
        else
        if ((fig == 1) && (pb->x > p3->x))
            dive41 (p1,p2,p3,p4,pb,p1->x,pa->y,pb->x,c,p1->y,fig);
        else
        if (fig == 6)
```

```
        dive30 (p1,p2,p3,p4,pb,p1->x,pa->y,pb->x,c,p1->y,fig,t);
}

/* funcion auxiliar a t1t6hor que supone al extremo inicial del obstaculo */
/* interior y el final exterior al cuadrilatero */
ainbdeh (p1,p2,p3,p4,pa,pb,c1,c2,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int c1,c2,fig,t;
{
        if ((fig == 1) && (pa->x <= p3->x))
            dive31 (p1,p2,p3,p4,pa,pa->x,p3->y,c2,pb->y,p1->y,fig,t);
        else
        if ((fig == 1) && (pa->x > p3->x))
            dive42 (p1,p2,p3,p4,pa,pa->x,c1,c2,pb->y,p1->y,fig);
        else
        if (fig == 6)
            dive31 (p1,p2,p3,p4,pa,pa->x,c1,c2,pb->y,p1->y,fig,t);
}

/* funcion auxiliar a t1t6hor que supone a los extremos inicial y final del *
/* obstaculo interiores al cuadrilatero */
ainbinh (p1,p2,p3,p4,pa,pb,c1,c2,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int c1,c2,fig,t;
{
        if ((fig == 1) && (pb->x <= p3->x))
            dive40 (p1,p2,p3,p4,pa,pb,p3->y,p3->y,p1->y,p1->y,fig,t);
        else
        if ((fig == 1) && (pb->x > p3->x))
            dive50 (p1,p2,p3,p4,pa,pb,p3->y,c2,p1->y,p1->y,fig);
        else
        if (fig == 6)
            dive43 (p1,p2,p3,p4,pa,pb,c1,c2,p1->y,p1->y);
}

/* funcion que divide un trapecio rectangulo derecho por un obstaculo */
/* horizontal */
tra2hor (p1,p2,p3,p4,pa,pb,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int fig,t;
{
    int ds,c1,c2,c3;

    ds = pa->y - pa->x + p1->x - p1->y;
    c1 = pa->y + p1->x - p1->y ; c2 = pb->x + p1->y - p1->x;
    c3 = pa->x + p1->y - p1->x;
    if ((ds >= 0) && (pb->x >= p3->x))
        dive20 (p1,p2,p3,p4,c1,pa->y,p3->x,pa->y,t);
    else
    if ((ds >= 0) && (pb->x < p3->x))
        aebi2h (p1,p2,p3,p4,pa,pb,c1,c2,fig,t);
    else
    if ((ds < 0) && (pb->x >= p3->x))
        aibe2h (p1,p2,p3,p4,pa,pb,c3,fig,t);
    else
```

```
    if ((ds < 0) && (pb->x < p3->x))
        if (pa->x >= p2->x)
            dive40 (p1,p2,p3,p4,pa,pb,p3->y,p3->y,p1->y,p1->y,fig,t);
        else
        if (pa->x < p2->x)
            dive51 (p1,p2,p3,p4,pa,pb,c3,p3->y,p1->y,p1->y,fig);
}

/* funcion auxiliar a tra2hor que considera al extremo inicial del obstaculo
/* exterior y al final interior al trapecio */
aebi2h (p1,p2,p3,p4,pa,pb,c1,c2,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int c1,c2,fig,t;
{
    if (pb->x >= p2->x)
        dive30 (p1,p2,p3,p4,pb,c1,pa->y,pb->x,p3->y,p1->y,fig,t);
    else
    if (pb->x < p2->x)
        dive44 (p1,p2,p3,p4,pb,c1,pa->y,pb->x,c2,p1->y,fig);
}

/* funcion auxiliar a tra2hor que considera al extremo inicial del obstaculo
/* interior y al final exterior al trapecio */
aibe2h (p1,p2,p3,p4,pa,pb,c,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int c,fig,t;
{
    if (pa->x >= p2->x)
        dive31 (p1,p2,p3,p4,pa,pa->x,p3->y,p3->x,pb->y,p1->y,fig,t);
    else
    if (pa->x < p2->x)
        dive45 (p1,p2,p3,p4,pa,pa->x,c,p3->x,pb->y,p1->y,fig);
}

/* funcion que divide un trapecio isosceles o un triangulo rectangulo */
/* isosceles por un obstaculo horizontal */
t3t7hor (p1,p2,p3,p4,pa,pb,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int fig,t;
{
    int dsi,dsf,c1,c2,c3,c4,c5,c6;

    dsi = pa->y - pa->x + p1->x - p1->y;
    dsf = pb->x + pb->y - p4->x - p4->y;
    c1 = pa->y + p1->x - p1->y ; c3 = pa->x + p1->y - p1->x;
    c2 = p4->x + p4->y - pb->y ; c4 = p4->x + p4->y - pb->x;
    c5 = pb->x + p1->y - p1->x ; c6 = p4->x + p4->y - pa->x;
    if ((dsi >= 0) && (dsf >= 0))
        aebe37h (p1,p2,p3,p4,pa,c1,c2,fig,t);
    else
    if ((dsi >= 0) && (dsf < 0))
        aebi37h (p1,p2,p3,p4,pa,pb,c1,c4,c5,fig,t);
    else
    if ((dsi < 0) && (dsf >= 0))
        aibe37h (p1,p2,p3,p4,pa,pb,c2,c3,c6,fig,t);
```

```
    else
    if ((dsi < 0) && (dsf < 0))
      if ((fig == 3) && ((pa->x < p2->x) && (pb->x >= p3->x)))
          dive52 (p1,p2,p3,p4,pa,pb,c3,c4,p1->y,p1->y,fig) ;
      else
      if ((fig == 3) && (pa->x < p2->x) && (pb->x <= p3->x))
          dive51 (p1,p2,p3,p4,pa,pb,c3,p2->y,p1->y,p1->y,fig);
      else
      if ((fig == 3) && (pa->x >= p2->x) && (pb->x > p3->x))
          dive50 (p1,p2,p3,p4,pa,pb,p2->y,c4,p1->y,p1->y,fig);
      else
      if (fig == 3)
          dive40 (p1,p2,p3,p4,pa,pb,p2->y,p2->y,p1->y,p1->y,fig,t);
      else
      if (fig == 7)
          dive52 (p1,p2,p3,p4,pa,pb,c3,c4,p1->y,p1->y,fig) ;
}

/* funcion auxiliar a t3t7hor que considera a los extremos del obstaculo */
/* exteriores a la figura */
aebe37h (p1,p2,p3,p4,pa,c1,c2,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa;
int c1,c2,fig,t;
{
    if (fig == 3)
        dive20 (p1,p2,p3,p4,c1,pa->y,c2,pa->y,t);
    else
    if (fig == 7)
        dive21 (p1,p2,p3,p4,c1,pa->y,c2,pa->y,t);
}

/* funcion auxiliar a t3t7hor que considera al extremo inicial del obstaculo */
/* exterior y al final interior a la figura */
aebi37h (p1,p2,p3,p4,pa,pb,c1,c2,c3,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int c1,c2,c3,fig,t;
{
        if (((fig == 3) && ((pb->x <= p3->x) && (pb->x >= p2->x))) ||
            ((fig == 7) && (pb->x == p2->x)))
            dive30 (p1,p2,p3,p4,pb,c1,pa->y,pb->x,p2->y,p1->y,fig,t);
        else
        if (((fig == 3) && (pb->x > p3->x)) ||
            ((fig == 7) && (pb->x > p2->x)))
            dive41 (p1,p2,p3,p4,pb,c1,pa->y,pb->x,c2,p1->y,fig);
        else
        if (((fig == 3) && (pb->x < p2->x)) ||
            ((fig == 7) && (pb->x < p2->x)))
            dive44 (p1,p2,p3,p4,pb,c1,pa->y,pb->x,c3,p1->y,fig);
}

/* funcion auxiliar a t3t7hor que considera al extremo inicial del obstaculo */
/* interior y al final exterior a la figura */
aibe37h (p1,p2,p3,p4,pa,pb,c1,c2,c3,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int c1,c2,c3,fig,t;
```

```
{
        if (((fig == 3) && ((pa->x <= p3->x) && (pa->x >= p2->x))) ||
            ((fig == 7) && (pa->x == p2->x)))
            dive31 (p1,p2,p3,p4,pa,pa->x,p2->y,c1,pb->y,p1->y,fig,t);
        else
        if (((fig == 3) && (pa->x < p2->x)) ||
            ((fig == 7) && (pa->x < p2->x)))
            dive45 (p1,p2,p3,p4,pa,pa->x,c2,c1,pb->y,p1->y,fig);
        else
        if (((fig == 3) && (pa->x > p3->x)) ||
            ((fig == 7) && (pa->x > p2->x)))
            dive42 (p1,p2,p3,p4,pa,pa->x,c3,c1,pb->y,p1->y,fig);
}

/* funcion que divide un romboide agudo por un obstaculo horizontal */
rom4hor (p1,p2,p3,p4,pa,pb,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int fig,t;
{
    int dsi,dsf,c1,c2;

    dsi = pa->y - pa->x + p1->x - p1->y;
    dsf = pb->y - pb->x + p4->x - p4->y;
    c1 = pa->y + p1->x - p1->y ; c2 = pb->y + p4->x - p4->y;
    if ((dsi >= 0) && (dsf <= 0))
        dive20 (p1,p2,p3,p4,c1,pa->y,c2,pa->y,t);
    else
    if ((dsi >= 0) && (dsf > 0))
        dive30 (p1,p2,p3,p4,pb,c1,pa->y,pb->x,p3->y,p1->y,fig,t);
    else
    if ((dsi < 0) && (dsf <= 0))
         dive31 (p1,p2,p3,p4,pa,pa->x,p3->y,c2,pa->y,p1->y,fig,t);
    else
    if ((dsi < 0) && (dsf > 0))
        dive40 (p1,p2,p3,p4,pa,pb,p3->y,p3->y,p1->y,p1->y,fig,t) ;
}

/* funcion que divide un romboide obtuso por un obstaculo horizontal */
rom5hor (p1,p2,p3,p4,pa,pb,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int fig,t;
{
    int dsi,dsf,c1,c2;

    dsi = pa->x + pa->y - p1->x - p1->y;
    dsf = pb->x + pb->y - p4->x - p4->y;
    c1 = p1->x + p1->y - pa->y ; c2 = p4->x + p4->y - pb->y;
    if ((dsi <= 0) && (dsf >= 0))
        dive20 (p1,p2,p3,p4,c1,pa->y,c2,pa->y,t);
    else
    if ((dsi <= 0) && (dsf < 0))
        dive30 (p1,p2,p3,p4,pb,c1,pa->y,pb->x,p3->y,p1->y,fig,t);
    else
    if ((dsi > 0) && (dsf >= 0))
        dive31 (p1,p2,p3,p4,pa,pa->x,p3->y,c2,pa->y,p1->y,fig,t);
```

```
    else
    if ((dsi > 0) && (dsf < 0))
        dive40 (p1,p2,p3,p4,pa,pb,p3->y,p3->y,p1->y,p1->y,fig,t)
}

/* funcion que considera la presencia de obstaculos verticales */
obsver (cuad,seg)
struct cuadrilatero *cuad;
struct segmento *seg;
{
    struct cuadrilatero cuar;
    struct segmento segr;
    struct punto p1,p2,p3,p4,pi,pf;
    int fig,por,po,t,tr,tx,ty,cx,cy;

    po = cuad->pos ; t = seg->tipo;
    switch (po) {
    case 0 : {
        desco1 (cuad,&p1,&p2,&p3,&p4,&fig,&po);
        desco2 (seg,&pi,&pf,&t);
        switch (fig) {
        case 0 :
            recver (&p1,&p2,&p3,&p4,&pi,&pf,fig,t);
            break;
        case 1 : case 6 :
            if (pi.y < pf.y)
                t1t6ver (&p1,&p2,&p3,&p4,&pi,&pf,fig,t);
            else
                t1t6ver (&p1,&p2,&p3,&p4,&pf,&pi,fig,t);
            break;
        case 2 :
            if (pi.y < pf.y)
                tra2ver (&p1,&p2,&p3,&p4,&pi,&pf,fig,t);
            else
                tra2ver (&p1,&p2,&p3,&p4,&pf,&pi,fig,t);
            break;
        case 3 : case 7 :
            if (pi.y < pf.y)
                t3t7ver (&p1,&p2,&p3,&p4,&pi,&pf,fig,t);
            else
                t3t7ver (&p1,&p2,&p3,&p4,&pf,&pi,fig,t);
            break;
        case 4 :
            if (pi.y < pf.y)
                rom4ver (&p1,&p2,&p3,&p4,&pi,&pf,fig,t);
            else
                rom4ver (&p1,&p2,&p3,&p4,&pf,&pi,fig,t);
            break;
        case 5 :
            if (pi.y < pf.y)
                rom5ver (&p1,&p2,&p3,&p4,&pi,&pf,fig,t);
            else
                rom5ver (&p1,&p2,&p3,&p4,&pf,&pi,fig,t);
            break;
        }
```

```
    break;
    case 1 : case 2 : case 3 : {
        constrans (cuad,&tx,&ty);
        consrot (cuad,&cx,&cy);
        rotdir (&cuar,cuad,tx,ty,cx,cy);
        rtdirs (&segr,seg,po,tx,ty,cx,cy);
        desco1 (&cuar,&p1,&p2,&p3,&p4,&fig,&por);
        desco2 (&segr,&pi,&pf,&tr);
        switch (tr) {
        case 0 :
            obshor (&cuar,&segr);
            break;
        case 1 :
            switch (fig) {
            case 0 :
                break;
            case 1 : case 6 :
                tit6ver (&p1,&p2,&p3,&p4,&pf,&pi,fig,t);
            break;
            case 2 :
                tra2ver (&p1,&p2,&p3,&p4,&pf,&pi,fig,t);
            break;
            case 3 : case 7 :
                t3t7ver (&p1,&p2,&p3.&p4,&pf,&pi,fig,t);
            break;
            case 4 :
                rom4ver (&p1,&p2,&p3,&p4,&pf,&pi,fig,t);
                break;
            case 5 :
                rom5ver (&p1,&p2,&p3,&p4,&pf,&pi,fig,t);
                break;
            }
        break;
        case 2 : case 3 :
        break;
        }
        constrans (&cuar,&tx,&ty) ; consrot (&cuar,&cx,&cy);
        invierte (po,tx,ty,cx,cy);
    }
    break;
    }
}

/* funcion que divide un rectangulo por un obstaculo vertical */
recver (p1,p2,p3,p4,pa,pb,fig,t)
struct punto *p1,*p2,*p3,*p4.*pa,*pb;
int fig,t;
{

    if ((pa->y <= p1->y) && (pb->y >= p3->y))
        dive22 (p1,p2,p3,p4,pa->x,p1->y,pa->x,p3->y,fig,t);
    else
    if ((pa->y <= p1->y) && (pb->y < p3->y))
        dive32 (p1,p2,p3,p4,pb,pa->x,pb->y,p1->x,p3->x,fig,t);
```

```
    else
    if ((pa->y > p1->y) && (pb->y >= p3->y))
        dive33 (p1,p2,p3,p4,pa,pa->y,p1->x,pa->x,p3->x,fig,t);
    else
    if ((pa->y > p1->y) && (pb->y < p3->y))
        dive46 (p1,p2,p3,p4,pa,pb,p1->x,p1->x,p3->x,p3->x,fig,t) ;


/* funcion que divide un trapecio rectangulo izquierdo o un triangulo */
/* rectangulo izquierdo por un obstaculo vertical */
t1t6ver (p1,p2,p3,p4,pa,pb,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int fig,t;
{
    int ds,c1,c2,yd;

    ds = pb->y + pb->x - p4->y - p4->x;
    c1 = p4->y + p4->x - pb->y ; c2 = p4->x + p4->y - pa->y;
    yd = p4->x + p4->y - pa->x;
    if ((pa->y <= p1->y) && ((ds >= 0) || (pb->y >= p2->y)))
        aebe16v (p1,p2,p3,p4,pa,pb,yd,fig,t);
    else
    if (((pa->y <= p1->y) && (ds < 0)) ||
        ((pa->y <= p1->y) && (pb->y < p2->y)))
        dive32 (p1,p2,p3,p4,pb,pa->x,pb->y,p1->x,c1,fig,t);
    else
    if ((pa->y > p1->y) && ((pb->y >= p2->y) || (ds >= 0)))
        aibe16v (p1,p2,p3,p4,pa,pb,c2,yd,fig,t);
    else
    if ((pa->y > p1->y) && ((pb->y < p2->y) || (ds < 0)))
        dive46 (p1,p2,p3,p4,pa,pb,p1->x,p1->x,c1,c2,fig,t);
}

/* funcion auxiliar a t1t6ver que considera a los extremos del obstaculo */
/* exteriores a la figura */
aebe16v (p1,p2,p3,p4,pa,pb,yd,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int yd,fig,t;
{
        if ((fig == 1) && (pb->x > p3->x))
            dive34 (p1,p2,p3,p4,pb,pa->x,yd,p1->x,pa->x,fig,t);
        else
            dive22 (p1,p2,p3,p4,pa->x,p1->y,pa->x,p3->y,fig,t);
}

/* funcion auxiliar a t1t6ver que considera al extremo inicial del obstaculo
/* interior y al final exterior a la figura */
aibe16v (p1,p2,p3,p4,pa,pb,c,yd,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int c,yd,fig,t;
{
      if ((fig == 1) && (pb->x > p3->x))
          dive47 (p1,p2,p3,p4,pa,pa->y,yd,p1->x,p1->x,pb->x,c,fig,t);
      else
      if (((fig == 1) && (pb->x <= p3->x)) || (fig == 6))
```

```
            dive33 (p1,p2,p3,p4,pa,pa->y,p1->x,pa->x,c,fig,t);
}

/* funcion que divide un trapecio rectangulo derecho por un obstaculo */
/* vertical */
tra2ver (p1,p2,p3,p4,pa,pb,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int fig,t;
{
    int ds,c1,c2,yd;

    yd = pa->x + p1->y - p1->x ; ds = pb->y - pb->x + p1->x - p1->y;
    c1 = pb->y + p1->x - p1->y ; c2 = pa->y + p1->x - p1->y;
    if ((pa->y <= p1->y) && ((ds >= 0) || (pb->y >= p3->y)))
        aebe2v (p1,p2,p3,p4,pa,pb,yd,fig,t);
    else
    if (((pa->y <= p1->y) && (ds < 0)) ||
        ((pa->y <= p1->y) && (pb->y < p3->y)))
        dive32 (p1,p2,p3,p4,pb,pa->x,pb->y,c1,p3->x,fig,t);
    else
    if ((pa->y > p1->y) && ((pb->y >= p3->y) || (ds >= 0)))
        aibe2v (p1,p2,p3,p4,pa,pb,c2,yd,fig,t);
    else
    if ((pa->y > p1->y) && ((pb->y < p3->y) || (ds < 0)))
        dive46 (p1,p2,p3,p4,pa,pb,c2,c1,p3->x,p3->x,fig,t);
}

/* funcion auxiliar a tra2ver que considera a los extremos del obstaculo */
/* exteriores a la figura */
aebe2v (p1,p2,p3,p4,pa,pb,yd,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int yd,fig,t;
{
        if (pb->x < p2->x)
            dive35 (p1,p2,p3,p4,pb,pa->x,yd,pa->x,p3->x,fig,t);
        else
            dive22 (p1,p2,p3,p4,pa->x,p1->y,pa->x,p3->y,fig,t);
}

/* funcion auxiliar a tra2ver que considera al extremo inicial del obstaculo
/* interior y al final exterior a la figura */
aibe2v (p1,p2,p3,p4,pa,pb,c,yd,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int c,yd,fig,t;
{
        if (pb->x >= p2->x)
            dive33 (p1,p2,p3,p4,pa,pa->y,c,pa->x,p3->x,fig,t);
        else
            dive48 (p1,p2,p3,p4,pa,pa->y,yd,c,pa->x,p3->x,p3->x,fig,t);
}

/* funcion que divide un trapecio isosceles o un triangulo rectangulo */
/* isosceles por un obstaculo vertical */
t3t7ver (p1,p2,p3,p4,pa,pb,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
```

```
int fig,t;
{
    int ds,ds1,c1,c2,c3,yi,yd;

    ds = pb->y - pb->x + p1->x - p1->y;
    ds1 = pb->y + pb->x - p4->y - p4->x ; yi = pa->x + p1->y - p1->x;
    c1 = pb->y + p1->x - p1->y ; c2 = p4->x + p4->y - pb->y;
    c3 = p4->x - pb->x + p1->x ; yd = p4->x + p4->y - pa->x;
    if ((pa->y <= p1->y) && (ds >= 0) && (pb->x < p2->x))
        dive35 (p1,p2,p3,p4,pb,pa->x,yi,pa->x,c3,fig,t);
    else
    if (((pa->y <= p1->y) && (ds < 0)) ||
        ((pa->y <= p1->y) && (ds1 < 0)) ||
        ((pa->y <= p1->y) && (pb->y < p2->y)))
        dive32 (p1,p2,p3,p4,pb,pa->x,pb->y,c1,c2,fig,t);
    else
    if ((fig == 3) && (pa->y <= p1->y) && (pb->y >= p3->y)
        && (pb->x >= p2->x) && (pb->x <= p3->x))
        dive22 (p1,p2,p3,p4,pa->x,p1->y,pa->x,p3->y,fig,t);
    else
    if ((fig == 7) && (pa->y <= p1->y) && (ds >= 0)
        && (ds1 >= 0) && (pb->x == p2->x))
        dive22 (p1,p2,p3,p4,pa->x,p1->y,pa->x,p2->y,fig,t);
    else
    if (((pa->y <= p1->y) && (ds1 >= 0)) &&
       (((fig == 3) && (pb->x > p3->x)) ||
       ((fig == 7) && (pb->x > p2->x))))
        dive34 (p1,p2,p3,p4,pb,pa->x,yd,c3,pb->x,fig,t);
    else
    if (pa->y > p1->y)
        aibe37v (p1,p2,p3,p4,pa,pb,c3,yi,yd,ds,ds1,fig,t);
}

/* funcion auxiliar a t3t7ver que considera al extremo inicial del obstaculo
/* interior y al extremo final exterior a la figura */
aibe37v (p1,p2,p3,p4,pa,pb,c,yi,yd,ds,ds1,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int c,yi,yd,ds,ds1,fig,t;
{
    int c1,c2,c3,c4;

    c1 = pa->y + p1->x - p1->y ; c2 = p4->x + p4->y - pa->y;
    c3 = pb->y + p1->x - p1->y ; c4 = p4->x + p4->y - pb->y;
    if ((ds >= 0) && (pb->x < p2->x))
        dive48 (p1,p2,p3,p4,pa,pa->y,yi,c1,pb->x,c,c2,fig,t);
    else
    if ((pb->y > p2->y) && (((fig == 7) && (pb->x == p2->x)) ||
       ((fig == 3) && (pb->x >= p2->x) && (pb->x <= p3->x))))
        dive33 (p1,p2,p3,p4,pa,pa->y,c1,pa->x,c2,fig,t);
    else
    if ((ds1 >= 0) && (((fig == 3) && (pb->x > p3->x)) ||
        ((fig == 7) && (pb->x > p2->x))))
        dive47 (p1,p2,p3,p4,pa,pa->y,yd,c1,c,pb->x,c2,fig,t);
    else
    if ((pb->y < p2->y) || (ds < 0) || (ds1 < 0))
```

```
        dive46 (p1,p2,p3,p4,pa,pb,c1,c3,c4,c2,fig,t);
}

/* funcion que divide un romboide agudo por un obstaculo vertical */
rom4ver (p1,p2,p3,p4,pa,pb,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;        -!
int fig,t;
{
    int dsi,dsf,c1,c2,c3,c4,yd;

    dsf = pb->y - pb->x + p1->x - p1->y;
    dsi = pa->y - pa->x + p4->x - p4->y;
    c1 = pb->y + p1->x - p1->y ; c2 = pb->y + p4->x - p4->y;
    c3 = p4->x + pb->x - p1->x ; c4 = p1->x + pb->x - p4->x;
    if ((pa->y <= p1->y) && (dsf >= 0) && (pb->x < p2->x) && (pb->x < p4->x))
        yd = pa->x + p1->y - p1->x;
        dive35 (p1,p2,p3,p4,pb,pa->x,yd,pa->x,c3,fig,t);
    }
    else
    if (((pa->y <= p1->y) && (dsf < 0)) ||
        ((pa->y <= p1->y) && (pb->y < p3->y)))
         dive32 (p1,p2,p3,p4,pb,pa->x,pb->y,c1,c2,fig,t);
    else
    if ((pa->y <= p1->y) && (pb->y >= p3->y)
        && (pb->x >= p2->x) && (pb->x <= p4->x))
        dive22 (p1,p2,p3,p4,pa->x,p1->y,pa->x,p3->y,fig,t);
    else
    if ((dsi <= 0) && (pb->y >= p3->y) && (pb->x > p4->x) && (pb->x > p2->x))
        yd = pa->x + p4->y - p4->x;
        dive36 (p1,p2,p3,p4,pa,yd,c4,pb->x,pb->x);
    }
    else
    if (((dsi > 0) && (pb->y >= p3->y)) ||
        ((pa->y > p1->y) && (pb->y >= p3->y))) {
        c3 = pa->y + p1->x - p1->y;
        c4 = pa->y + p4->x - p4->y;
        dive33 (p1,p2,p3,p4,pa,pa->y,c3,pa->x,c4,fig,t);
    }
    else
    if ((pa->y > p1->y) && (pb->y < p3->y) && (dsf < 0) && (dsi > 0))
        dive46 (p1,p2,p3,p4,pa,pb,c3,c1,c2,c4,fig,t);
}

/* funcion que divide un romboide obtuso por un obstaculo vertical */
rom5ver (p1,p2,p3,p4,pa,pb,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int fig,t;
{
    int dsi,dsf,c1,c2,c3,c4,yd;

    dsf = pb->y + pb->x - p4->x - p4->y;
    dsi = pa->y + pa->x - p1->x - p1->y;
    c1 = p1->x + p1->y - pb->y ; c2 = p4->x + p4->y - pb->y;
    c3 = p4->x + pb->x - p1->x ; c4 = p1->x + pb->x - p4->x;
    if ((pa->y <= p1->y) && (dsf >= 0) && (pb->x > p3->x) && (pb->x > p1->x))
```

```
    yd = p4->y + p4->x - pa->x;
    dive34 (p1,p2,p3,p4,pb,pa->x,yd,c4,pb->x,fig,t);
}
else
if (((pa->y <= p1->y) && (dsf < 0)) ||
    ((pa->y <= p1->y) && (pb->y < p3->y)))
    dive32 (p1,p2,p3,p4,pb,pa->x,pb->y,c1,c2,fig,t);
else
if ((pa->y <= p1->y) && (pb->y >= p3->y)
    && (pb->x >= p2->x) && (pb->x <= p3->x))
    dive22 (p1,p2,p3,p4,pa->x,p1->y,pa->x,p3->y,fig,t);
else
if ((dsi <= 0) && (pb->y >= p3->y) && (pb->x < p3->x) && (pb->x < p1->x)
    yd = p1->y + p1->x - pa->x;
    dive37 (p1,p2,p3,p4,pa,yd,pb->x,pb->x,c3);
}
else
if (((dsi > 0) && (pb->y >= p3->y)) ||
    ((pa->y > p1->y) && (pb->y >= p3->y))) {
    c3 = p1->x + p1->y - pa->y;
    c4 = p4->x + p4->y - pa->y;
    dive33 (p1,p2,p3,p4,pa,pa->y,c3,pa->x,c4,fig,t);
}
else
if ((pa->y > p1->y) && (pb->y < p3->y) && (dsf < 0) && (dsi > 0))
    dive46 (p1,p2,p3,p4,pa,pb,c3,c1,c2,c4,fig,t);
```

```
/* funcion que considera la presencia de obstaculos colicuos a 45 grados */
obbli45 (cuad,seg)
struct cuadrilatero *cuad;
struct segmento *seg;
{
    struct cuadrilatero cuar;
    struct segmento segr;
    struct punto p1,p2,p3,p4,pi,pf;
    int fig,por,po,t,tr,tx,ty,cx,cy;

    po = cuad->pos ; t = seg->tipo;
    switch (po) {
    case 0 : {
        desco1 (cuad,&p1,&p2,&p3,&p4,&fig,&po);
        desco2 (seg,&pi,&pf,&t);
        switch (fig) {
        case 0 :
            recta45 (&p1,&p2,&p3,&p4,&pi,&pf,fig,t);
            break;
        case 1 : case 6 :
            if (pi.y < pf.y)
                t1t645 (&p1,&p2,&p3,&p4,&pi,&pf,fig,t);
            else
                t1t645 (&p1,&p2,&p3,&p4,&pf,&pi,fig,t);
            break;
        case 2 :
            if (pi.y < pf.y)
                tra245 (&p1,&p2,&p3,&p4,&pi,&pf,fig,t);
            else
                tra245 (&p1,&p2,&p3,&p4,&pf,&pi,fig,t);
            break;
        case 3 : case 7 :
            if (pi.y < pf.y)
                t3t745 (&p1,&p2,&p3,&p4,&pi,&pf,fig,t);
            else
                t3t745 (&p1,&p2,&p3,&p4,&pf,&pi,fig,t);
            break;
        case 4 :
            if (pi.y < pf.y)
                rom445 (&p1,&p2,&p3,&p4,&pi,&pf,fig,t);
            else
                rom445 (&p1,&p2,&p3,&p4,&pf,&pi,fig,t);
            break;
        case 5 :
            if (pi.y < pf.y)
                rom545 (&p1,&p2,&p3,&p4,&pi,&pf,fig,t);
            else
                rom545 (&p1,&p2,&p3,&p4,&pf,&pi,fig,t);
            break;
        }
    }
    break;
    case 1 : case 2 : case 3 : {
        constrans (cuad,&tx,&ty);
        consrot (cuad,&cx,&cy);
```

```
        rotdir (&cuar,cuad,tx,ty,cx,cy);
        rtdirs (&segr,seg,po,tx,ty,cx,cy);
        desco1 (&cuar,&p1,&p2,&p3,&p4,&fig,&per);
        desco2 (&segr,&pi,&pf,&tr);
        switch (tr) {
        case 0 : case 1 :
            break;
        case 2 :
            switch (fig) {
            case 0 :
                break;
            case 1 : case 6 :
                t1t645 (&p1,&p2,&p3,&p4,&pf,&pi,fig,t);
                break;
            case 2 :
                tra245 (&p1,&p2,&p3,&p4,&pf,&pi,fig,t);
                break;
            case 3 : case 7 :
                t3t745 (&p1,&p2,&p3,&p4,&pf,&pi,fig,t);
                break;
            case 4 :
                rom445 (&p1,&p2,&p3,&p4,&pf,&pi,fig,t);
                break;
            case 5 :
                rom545 (&p1,&p2,&p3,&p4,&pf,&pi,fig,t);
                break;
            }
            break;
        case 3 :
            obbli135 (&cuar,&segr);
            break;
        }
        constrans (&cuar,&tx,&ty) ; consrot (&cuar,&cx,&cy);
        invierte (po,tx,ty,cx,cy);
    }
    break;
    }


/* funcion que divide un rectangulo por un obstaculo oblicuo a 45 grados */
recta45 (p1,p2,p3,p4,pa,pb,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int fig,t;
{
    int x5,x6,y5,y6;

    y5 = p1->x + pa->y - pa->x ; y6 = p3->x + pa->y - pa->x ;
    x5 = p1->y - pa->y + pa->x ; x6 = p3->y - pa->y + pa->x ;
    if ((pa->x > p1->x) && (pa->y > p1->y) && (pb->x < p3->x) &&
        (pb->y < p3->y))
        dive46 (p1,p2,p3,p4,pa,pb,p1->x,p1->x,p3->x,p3->x,fig,t);
    else
    if ((y5 > p1->y) && (y5 < p3->y))
        laizqr (p1,p2,p3,p4,pa,pb,y5,fig,t);
    else
```

```
    if (y5 == p1->y)
        pinizr (p1,p2,p3,p4,pa,pb,y5,fig,t);
    else
    if ((x5 > p1->x) && (x5 < p3->x))
        lainfr (p1,p2,p3,p4,pa,pb,x5,fig,t);
}

/* funcion que considera al obstaculo incidente por el lado izquierdo */
/* del rectangulo */
laizqr (p1,p2,p3,p4,pa,pb,y5,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int y5,fig,t;
{
    int x6,y6;

    y6 = p3->x + pa->y - pa->x ; x6 = p3->y - pa->y + pa->x ;
    if ((y6 < p3->y) || (y6 == p3->y))
        aebe045 (p1,p2,p3,p4,pa,pb,y5,y6,fig,t);
    else
    if (x6 < p3->x)
        if ((pa->x <= p1->x) && (pb->y >= p3->y))
            dive30 (p1,p2,p3,p4,pb,p1->x,y5,x6,p3->y,p1->y,fig,t);
        else
            ayuda1 (p1,p2,p3,p4,pa,pb,x6,y5,fig,t);


/* funcion que considera al obstaculo coincidente con el vertice inf-izq */
pinizr (p1,p2,p3,p4,pa,pb,y5,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int y5,fig,t;
{
    int x6,y6;

    y6 = p3->x + pa->y - pa->x ; x6 = p3->y - pa->y + pa->x ;
    if ((y6 < p3->y) || (y6 == p3->y))
        aebe045 (p1,p2,p3,p4,pa,pb,y5,y6,fig,t);
    else
    if (x6 < p3->x)
        if ((pa->x <= p1->x) && (pb->y >= p3->y))
            dive22 (p1,p2,p3,p4,p1->x,y5,x6,p3->y,fig,t);
        else
            ayuda1 (p1,p2,p3,p4,pa,pb,x6,y5,fig,t);
}

/* funcion auxiliar a ladizq y pinizq que considera a los extremos del */
/* obstaculo exteriores a la figura */
aebe045 (p1,p2,p3,p4,pa,pb,y5,y6,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int y5,y6,fig,t;
{
        if ((pa->x <= p1->x) && (pb->x >= p3->x))
            dive20 (p1,p2,p3,p4,p1->x,y5,p3->x,y6,t);
        else
            ayuda0 (p1,p2,p3,p4,pa,pb,y5,y6,fig,t);
```

```
/* funcion auxiliar a laizqr y pinizr */
ayuda0 (p1,p2,p3,p4,pa,pb,y5,y6,fg,ti)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int y5,y6;
{                                                    -!
    if ((pa->x <= p1->x) && (pb->x < p3->x))
        dive30 (p1,p2,p3,p4,pb,p1->x,y5,pb->x,p3->y,p1->y,fg,ti);
    else
    if ((pa->x > p1->x) && (pb->x >= p3->x))
        dive31 (p1,p2,p3,p4,pa,pa->x,p3->y,p3->x,y6,p1->y,fg,ti);
}

/* funcion auxiliar a laizqr y pinizr */
ayuda1 (p1,p2,p3,p4,pa,pb,x6,y5,fg,ti)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int x6,y5,fg,ti;
{
    if ((pa->x <= p1->x) && (pb->y < p3->y))
        dive30 (p1,p2,p3,p4,pb,p1->x,y5,pb->x,p3->y,p1->y,fg,ti);
    else
    if ((pa->x > p1->x) && (pb->y >= p3->y))
        dive33 (p1,p2,p3,p4,pa,pa->y,p1->x,x6,p3->x,fg,ti);
}

/* funcion que considera al obstaculo incidente por el lado inferior del */
/* rectangulo */
lainfr (p1,p2,p3,p4,pa,pb,x5,fig,t)
truct punto *p1,*p2,*p3,*p4,*pa,*pb;
nt x5,fig,t;
{
    int x6,y6;

    y6 = p3->x + pa->y - pa->x ; x6 = p3->y - pa->y + pa->x ;
    if ((y6 < p3->y) || (y6 == p3->y))
        yin45h (p1,p2,p3,p4,pa,pb,x5,y6,fig,t);
    else
    if (x6 < p3->x)
        if ((pa->y <= p1->y) && (pb->y >= p3->y))
          dive22 (p1,p2,p3,p4,x5,p1->y,x6,p3->y,fig,t);
        else
        if ((pa->y <= p1->y) && (pb->y < p3->y))
          dive32 (p1,p2,p3,p4,pb,x5,pb->y,p1->x,p3->x,fig,t);
        else
        if ((pa->y > p1->y) && (pb->y >= p3->y))
          dive33 (p1,p2,p3,p4,pa,pa->y,p1->x,x6,p3->x,fig,t);
}

/* funcion auxiliar a lainfr que considera al obstaculo comprendido entre */
/* las ordenadas maxima y minima de la figura */
yin45h (p1,p2,p3,p4,pa,pb,x5,y6,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int x5,y6,fig,t;
{
    if ((pa->y <= p1->y) && (pb->x >= p3->x))
```

```
        aibs045 (p1,p2,p3,p4,pa,x5,y6,fig,t);
    else
    if ((pa->y <= p1->y) && (pb->x < p3->x))
        dive32 (p1,p2,p3,p4,pb,x5,pb->y,p1->x,p3->x,fig,t);
    else
    if  ((pa->y > p1->y) && (pb->x >= p3->x))
        dive31 (p1,p2,p3,p4,pa,pa->x,p3->y,p3->x,y6,p1->y,fig,t);
}

/* funcion auxiliar a lainfr que considera a los extremos del obstaculo */
/* exteriores a la figura */
aibs045 (p1,p2,p3,p4,pa,x5,y6,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa;
int x5,y6,fig,t;
{
    if (y6 < p3->y)
        dive31 (p1,p2,p3,p4,pa,x5,p3->y,p3->x,y6,p1->y,fig,t);
    else
    if (y6 == p3->y)
        dive22 (p1,p2,p3,p4,x5,p1->y,p3->x,y6,fig,t);
}

/* funcion que divide un trapecio rectangulo izquierdo o un triangulo */
/* rectangulo izquierdo por un obstaculo oblicuo a 45 grados */
tit645 (p1,p2,p3,p4,pa,pb,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int fig,t;
{
    int ds,x5,x6,y5,y6,xd,xd1;

    ds = pb->y + pb->x - p4->y - p4->x;
    y5 = p1->x + pa->y - pa->x ; x5 = p1->y - pa->y + pa->x ;
    xd = p4->y + p4->x - pa->y ; xd1 = p4->y + p4->x - pb->y;
    if (fig == 1) {
        x6 = p3->y - pa->y + pa->x ; y6 = p3->x + pa->y - pa->x;
    }
    if ((pa->x > p1->x) && (pa->y > p1->y) &&
       (((ds < 0) && (fig == 6)) ||
        ((ds < 0) && (fig == 1) && (pb->x > p3->x)) ||
        ((pb->y < p2->y) && (pb->x >= p2->x) && (pb->x <= p3->x))))
        dive46 (p1,p2,p3,p4,pa,pb,p1->x,p1->x,xd1,xd,fig,t);
    else
        if ((y5 > p1->y) && (y5 < p2->y))
            ladizq (p1,p2,p3,p4,pa,pb,y5,fig,t);
        else
        if (y5 == p1->y)
            pinizq (p1,p2,p3,p4,pa,pb,y5,fig,t);
        else
        if ((x5 > p1->x) && (x5 < p4->x))
            ladinft (p1,p2,p3,p4,pa,pb,x5,fig,t);
}

/* funcion que considera al obstaculo incidente por el lado izquierdo */
ladizq (p1,p2,p3,p4,pa,pb,y5,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
```

```
int y5,fig,t;
{
    int ds,x6,xd,xd1,y6;

    ds = pb->y + pb->x - p4->y - p4->x;
    y6 = (p4->x + p4->y + pa->y - pa->x) / 2; xd1 = p4->x + p4->y - y6;
    x6 = p3->y - pa->y + pa->x ; xd = p4->x + p4->y - y5;
    if ((y6 < p2->y) || (y6 == p2->y))
        yin1645 (p1,p2,p3,p4,pa,pb,xd,xd1,y5,y6,ds,fig,t);
    else
    if ((fig == 1) && (x6 < p3->x))
        if ((pa->x <= p1->x) && (pb->y >= p3->y))
            dive30 (p1,p2,p3,p4,pb,p1->x,y5,x6,p3->y,p1->y,fig,t);
        else
            help1 (p1,p2,p3,p4,pa,pb,y5,x6,fig,t);
}

/* funcion auxiliar a ladizq que considera al obstaculo comprendido entre */
/* las ordenadas maxima y minima de la figura */
yin1645 (p1,p2,p3,p4,pa,pb,xd,xd1,y5,y6,ds,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int xd,xd1,y5,y6,ds,fig,t;
{
    if ((pa->x <= p1->x) && (ds >= 0))
        aebe1645 (p1,p2,p3,p4,pa,xd,xd1,y5,y6,fig,t);
    else
    if ((pa->x <= p1->x) && (ds < 0))
        aebi1645 (p1,p2,p3,p4,pa,pb,y5,fig,t);
    else
    if ((pa->x > p1->x) && (ds >= 0))
        dive47 (p1,p2,p3,p4,pa,pa->y,y6,p1->x,p1->x,xd1,xd,fig,t);
}

/* funcion auxiliar a ladizq que considera los extremos del obstaculo */
/* exteriores a la figura */
aebe1645 (p1,p2,p3,p4,pa,xd,xd1,y5,y6,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa;
int xd,xd1,y5,y6,fig,t;
{
        if (fig == 1)
            dive47 (p1,p2,p3,p4,pa,y5,y6,p1->x,p1->x,xd1,xd,fig,t);
        else
        if (fig == 6)
            dive38 (p1,p2,p4,p1->x,y5,xd1,y6,xd);
}

/* funcion auxiliar a ladizq que considera el extremo inicial del obstaculo */
/* exterior y el final interior a la figura */
aebi1645 (p1,p2,p3,p4,pa,pb,y5,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int y5,fig,t;
{
    int ys;

    ys = p4->x + p4->y - pb->x;
```

```
    if ((fig == 1) && (pb->x > p3->x))
        dive41 (p1,p2,p3,p4,pb,p1->x,y5,pb->x,ys,p1->y,fig);
    else
        help0 (p1,p2,p3,p4,pa,pb,y5,fig,t);
}

/* funcion que considera al obstaculo coincidente con el punto inf. izq. */
pinizq (p1,p2,p3,p4,pa,pb,y5,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int y5,fig,t;
{
    int ds,x5,x6,xd,y6;

    ds = pb->y + pb->x - p4->y - p4->x;
    x6 = p3->y - pa->y + pa->x ; xd = p4->x + p4->y - pa->y;
    y6 = (p4->x + p4->y + pa->y - pa->x) / 2;
    x5 = (p4->x + p4->y + pa->x - pa->y) / 2;
    if ((y6 < p2->y) || (y6 == p2->y))
        piz1645 (p1,p2,p3,p4,pa,pb,x5,xd,y6,ds,fig,t);
    else
    if ((fig == 1) && (x6 < p3->x))
        if ((pa->x <= p1->x) && (pb->y >= p3->y))
            dive22 (p1,p2,p3,p4,p1->x,p1->y,x6,p3->y,fig,t);
        else
            help1 (p1,p2,p3,p4,pa,pb,p1->y,x6,fig,t);
}

/* funcion auxiliar a ladizq que considera al obstaculo comprendido entre */
/* las ordenadas maxima y minima de la figura */
piz1645 (p1,p2,p3,p4,pa,pb,x5,xd,y6,ds,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int x5,xd,y6,ds,fig,t;
{
    if ((pa->x <= p1->x) && (ds >= 0))
        pex1645 (p1,p2,p3,p4,pb,x5,y6,fig,t);
    else
    if ((pa->x <= p1->x) && (ds < 0))
        pei1645 (p1,p2,p3,p4,pa,pb,fig,t);
    else
    if ((pa->x > p1->x) && (ds >= 0))
        dive47 (p1,p2,p3,p4,pa,pa->y,y6,p1->x,p1->x,x5,xd,fig,t);
}

/* funcion auxiliar a pinizq que considera los extremos del obstaculo */
/* exteriores a la figura */
pex1645 (p1,p2,p3,p4,pb,x5,y6,fig,t)
struct punto *p1,*p2,*p3,*p4,*pb;
int x5,y6,fig,t;
{
    if (fig == 1)
        dive34 (p1,p2,p3,p4,pb,p1->x,y6,p1->x,x5,fig,t);
    else
    if (fig == 6)
        dive21 (p1,p2,p3,p4,p1->x,p1->y,x5,y6,t);
```

```
/* funcion auxiliar a pinizq que considera el extremo inicial del obstaculo
/* exterior y el final interior a la figura */
pei1645 (p1,p2,p3,p4,pa,pb,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int fig,t;
{
    int x5;

    x5 = p4->x + p4->y - pb->y;
    if ((fig == 1) && (pb->x > p3->x))
        dive32 (p1,p2,p3,p4,pb,p1->x,pb->y,p1->x,x5,fig,t);
    else
        help0 (p1,p2,p3,p4,pa,pb,p1->y,fig,t);
}

/* funcion auxiliar a ladizq y pinizq */
help0 (p1,p2,p3,p4,pa,pb,yi,fg,ti)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int yi,fg,ti;
{
    int ys;

    ys = p4->x + p4->y - pb->x;
    if ((fg == 1) && (pb->x <= p3->x))
        dive30 (p1,p2,p3,p4,pb,p1->x,yi,pb->x,p3->y,p1->y,fg,ti);
    else
    if (fg == 6)
        dive30 (p1,p2,p3,p4,pb,p1->x,yi,pb->x,ys,p1->y,fg,ti);
}

/* funcion auxiliar a ladizq y pinizq */
help1 (p1,p2,p3,p4,pa,pb,ys,x6,fg,ti)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int x6,ys,fg,ti;
{
    int x5;

    x5 = p4->x + p4->y - pa->y;
    if ((pa->x <= p1->x) && (pb->y < p3->y))
        dive30 (p1,p2,p3,p4,pb,p1->x,ys,pb->x,p3->y,p1->y,fg,ti);
    else
    if ((pa->x > p1->x) && (pb->y >= p3->y))
        dive33 (p1,p2,p3,p4,pa,pa->y,p1->x,x6,x5,fg,ti);
}

/* funcion que considera al obstaculo incidente por el lado inferior */
ladinft (p1,p2,p3,p4,pa,pb,x5,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int x5,fig,t;
{
    int ds,x6,y6,xd,xd1;

    ds = pb->y + pb->x - p4->y - p4->x;
    x6 = p3->y - pa->y + pa->x ;
```

```
    y6 = (p4->x + p4->y + pa->y - pa->x) / 2;
    xd = p4->x + p4->y - pb->y ; xd1 = p4->x + p4->y - pa->y;
    if ((fig == 1 ) && (x6 < p3->x))
      auxil1 (p1,p2,p3,p4,pa,pb,x5,p1->x,p1->x,x6,xd1,xd,fig,t);
    else
    if ((y6 < p2->y) || (y6 == p2->y)) {
        if ((pa->y <= p1->y) && (ds >= 0))
           exf1645 (p1,p2,p3,p4,pa,pb,x5,y6,fig,t);
        else
        if ((pa->y <= p1->y) && (ds < 0))
            dive32 (p1,p2,p3,p4,pb,x5,pb->y,p1->x,xd,fig,t);
        else
        if  ((pa->y > p1->y) && (ds >= 0))
            ief1645 (p1,p2,p3,p4,pa,xd1,y6,fig,t);
    }
}

/* funcion auxiliar a ladinft que considera los extremos del obstaculo */
/* exteriores a la figura */
exf1645 (p1,p2,p3,p4,pa,pb,x5,y6,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int x5,y6,fig,t;
{
    int xd;

    xd = (p4->x + p4->y + pa->x - pa->y) / 2;
    if (((fig == 1) && (y6 < p2->y)) || (fig == 6))
        dive34 (p1,p2,p3,p4,pb,x5,y6,p1->x,xd,fig,t);
    else
    if ((fig == 1) && (y6 == p2->y))
        dive22 (p1,p2,p3,p4,x5,p1->y,p3->x,y6,fig,t);
}

/* funcion auxiliar a ladinft que considera el extremo inicial del obstaculo
/* interior y el final exterior a la figura */
ief1645 (p1,p2,p3,p4,pa,xd1,y6,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa;
int xd1,y6,fig,t;
{
    int xd;

    xd = p4->x + p4->y - y6;
    if (((fig == 1) && (y6 < p2->y)) || (fig == 6))
        dive47 (p1,p2,p3,p4,pa,pa->y,y6,p1->x,p1->x,xd,xd1,fig,t);
    else
    if ((fig == 1) && (y6 == p2->y))
        dive33 (p1,p2,p3,p4,pa,pa->y,p1->x,p3->x,xd1,fig,t);
}

/* funcion que divide un trapecio rectangulo derecho por un */
/* obstaculo oblicuo a 45 grados */
tra245 (p1,p2,p3,p4,pa,pb,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int fig,t;
{
```

```
    int ds,x5,x6,y5,y6,xi,xi1;

    ds = pa->y - pa->x + p1->x - p1->y;
    xi = pa->y - p1->y + p1->x ; xi1 = pb->y - p1->y + p1->x;
    y5 = p1->x + pa->y - pa->x ; x5 = p1->y - pa->y + pa->x ;
    x6 = p3->y - pa->y + pa->x ; y6 = p3->x + pa->y - pa->x;
    if ((ds < 0) && (pa->y > p1->y) && (pb->y < p3->y) && (pb->x < p3->x))
      dive46 (p1,p2,p3,p4,pa,pb,xi,xi1,p3->x,p3->x,fig,t);
    else
    if ((x5 > p1->x) && (x5 < p3->x))
        if (x6 < p3->x)
            auxil1 (p1,p2,p3,p4,pa,pb,x5,xi,xi1,x6,p3->x,p3->x,fig,t);
        else
        if ((y6 < p3->y) || (y6 == p3->y)) {
            if ((pa->y <= p1->y) && (pb->x >= p3->x))
                aebe245 (p1,p2,p3,p4,pb,x5,y6,fig,t);
            else
            if ((pa->y <= p1->y) && (pb->x < p3->x))
                dive32 (p1,p2,p3,p4,pb,x5,pb->y,xi1,p3->x,fig,t);
            else
            if  ((pa->y > p1->y) && (pb->x >= p3->x))
                dive31 (p1,p2,p3,p4,pa,pa->x,p3->y,p3->x,y6,p1->y,fig,t);
        }


/* funcion auxiliar a ladinft que considera los extremos del obstaculo */
/* exteriores a la figura */
aebe245 (p1,p2,p3,p4,pb,x5,y6,fig,t)
struct punto *p1,*p2,*p3,*p4,*pb;
int x5,y6,fig,t;
{
    int xi;

    xi = y6 - p1->y + p1->x;
    if (y6 < p3->y)
        dive32 (p1,p2,p3,p4,pb,x5,y6,xi,p3->x,fig,t);
    else
    if (y6 == p3->y)
        dive22 (p1,p2,p3,p4,x5,p1->y,p3->x,y6,fig,t);
}

/* funcion que divide un trapecio isosceles o un triangulo */
/* rectangulo isosceles por un obstaculo oblicuo a 45 grados */
t3t745 (p1,p2,p3,p4,pa,pb,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int fig,t;
{
    int ds,x5,x6,y6,xi,xi1,yd,xd,xd1;

    ds = pb->y + pb->x - p4->x - p4->y;
    xi = pa->y - p1->y + p1->x ; xi1 = pb->y - p1->y + p1->x;
    xd = p4->x + p4->y - pa->y ; xd1 = p4->x + p4->y - pb->y;
    yd = (p4->x + p4->y + pa->y - pa->x) / 2;
    x5 = p1->y - pa->y + pa->x ;
    x6 = p2->y - pa->y + pa->x ; y6 = p3->x + pa->y - pa->x;
```

```
    if ((pa->y > p1->y) && (pa->x > p1->x) &&
       (((ds < 0) && (fig == 7)) ||
        ((ds < 0) && (fig == 3) && (pb->x > p3->x)) ||
        ((pb->y < p2->y) && (pb->x >= p2->x) && (pb->x <= p3->x))))
        dive46 (p1,p2,p3,p4,pa,pb,xi,xi1,xd1,xd,fig,t);
    else
    if ((x5 > p1->x) && (x5 < p4->x))
        if ((x6 < p3->x) && (fig == 3))
            auxil1 (p1,p2,p3,p4,pa,pb,x5,xi,xi1,x6,xd,xd1,fig,t);
        else
        if ((y6 < p2->y) || (y6 == p2->y)) {
            if ((pa->y <= p1->y) && (ds >= 0))
                aebe3745 (p1,p2,p3,p4,pb,x5,y6,yd,fig,t);
            else
            if ((pa->y <= p1->y) && (ds < 0))
                dive32 (p1,p2,p3,p4,pb,x5,pb->y,xi1,xd1,fig,t);
            else
            if  ((pa->y > p1->y) && (ds >= 0))
                aibe3745 (p1,p2,p3,p4,pa,xi,xd,yd,y6,fig,t);
        }
}

/* funcion auxiliar a t3t745 que considera los extremos del obstaculo */
/* exteriores a la figura */
aebe3745 (p1,p2,p3,p4,pb,x5,y6,yd,fig,t)
struct punto *p1,*p2,*p3,*p4,*pb;
int x5,y6,yd,fig,t;
{
    int xd,xi;

    xd = p4->x + p4->y - yd ; xi = yd - p1->y + p1->x;
    if (y6 < p2->y)
        dive34 (p1,p2,p3,p4,pb,x5,yd,xi,xd,fig,t);
    else
    if (y6 == p2->y)
        dive22 (p1,p2,p3,p4,x5,p1->y,p3->x,p3->y,fig,t);
}

/* funcion auxiliar a t3t745 que considera el extremo inicial del obstaculo *
/* interior y el final exterior a la figura */
aibe3745 (p1,p2,p3,p4,pa,xi,xd,yd,y6,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa;
int xi,xd,yd,y6,fig,t;
{
    int xd1,xi1;

    xd1 = p4->x + p4->y - yd ; xi1 = yd - p1->y + p1->x;
    if (y6 < p2->y)
        dive47 (p1,p2,p3,p4,pa,pa->y,yd,xi,xi1,xd1,xd,fig,t);
    else
    if (y6 == p2->y)
        dive33 (p1,p2,p3,p4,pa,pa->y,xi,p3->x,xd,fig,t);
}

/* funcion que divide un romboide agudo por un obstaculo oblicuo a 45 grados
```

```
rom445 (p1,p2,p3,p4,pa,pb,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int fig,t;
{
    int xi,xs,xi1,xi2,xd1,xd2;

    xi = p1->y - pa->y + pa->x ; xs = p3->y - pa->y + pa->x;
    xi2 = pb->y - p1->y + p1->x ; xd2 = pb->y - p4->y + p4->x;
    xi1 = pa->y - p1->y + p1->x ; xd1 = pa->y - p4->y + p4->x;
    if ((pa->y > p1->y) && (pb->y < p3->y))
        dive46 (p1,p2,p3,p4,pa,pb,xi1,xi2,xd2,xd1,fig,t);
    else
    auxil1 (p1,p2,p3,p4,pa,pb,xi,xi1,xi2,xs,xd1,xd2,fig,t);
}

/* funcion auxiliar para generar algunas particiones del obstaculo a 45 */
auxil1 (p1,p2,p3,p4,pa,pb,xi,xi1,xi2,xs,xd1,xd2,fg,ti)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int xi,xi1,xi2,xs,xd1,xd2,fg,ti;
{

    if ((pa->y <= p1->y) && (pb->y >= p3->y))
        dive22 (p1,p2,p3,p4,xi,p1->y,xs,p3->y,fg,ti);
    else
    if ((pa->y <= p1->y) && (pb->y < p3->y))
        dive32 (p1,p2,p3,p4,pb,xi,pb->y,xi2,xd2,fg,ti);
    else
    if ((pa->y > p1->y) && (pb->y >= p3->y))
        dive33 (p1,p2,p3,p4,pa,pa->y,xi1,xs,xd1,fg,ti);
}

/* funcion que divide un romboide obtuso por un obstaculo oblicuo a 45 grados */
rom545 (p1,p2,p3,p4,pa,pb,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int fig,t;
{
    int ds,di1,x5,x6,xi1,xi2,xd1,xd2,yda,xib,xdb;

    x5 = p1->y - pa->y + pa->x ; x6 = p3->y - pa->y + pa->x ;
    ds = pb->y + pb->x - p4->x - p4->y;
    di1 = pa->y + pa->x - p1->x - p1->y;
    xi1 = p1->x + p1->y - pa->y ; xi2 = p1->x + p1->y - pb->y;
    xd1 = p4->x + p4->y - pa->y ; xd2 = p4->x + p4->y - pb->y;
    yda = (p4->y + p4->x + pa->y - pa->x) / 2;
    xib = p1->x + p1->y - yda ; xdb = p4->x + p4->y - yda ;
    if ((di1 > 0) && (ds < 0) && (pa->y > p1->y) && (pb->y < p3->y))
        dive46 (p1,p2,p3,p4,pa,pb,xi1,xi2,xd2,xd1,fig,t);
    else
    if (x6 <= p3->x)
        xint545 (p1,p2,p3,p4,pa,pb,x5,x6,xi1,xd1,xi2,xd2,fig,t);
    else
    if ((x6 > p3->x) && (x5 < p1->x))
        xext545 (p1,p2,p3,p4,pa,pb,xib,xdb,xi1,xd1,xi2,xd2,yda,ds,fig,t);
    else
    if ((x5 > p1->x) && (x5 < p4->x))
```

```
        if ((pa->y <= p1->y) && (ds >= 0))
            dive34 (p1,p2,p3,p4,pb,x5,yda,xib,xdb,fig,t);
        else
        if ((pa->y <= p1->y) && (ds < 0))
            dive32 (p1,p2,p3,p4,pb,x5,pb->y,xi2,xd2,fig,t);
        else
        if ((pa->y > p1->y) && (ds >= 0))
            dive47 (p1,p2,p3,p4,pa,pa->y,yda,xi1,xib,xdb,xd1,fig,t);
}

/* funcion auxiliar a rom545 que considera al obstaculo interior a los */
/* limites de las x de la figura */
xint545 (p1,p2,p3,p4,pa,pb,x5,x6,xi1,xd1,xi2,xd2,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int x5,x6,xi1,xd1,xi2,xd2,fig,t;
{
    int xia,xda,yd,di1;

    yd = (p1->y + p1->x + pa->y - pa->x) / 2;
    xia = p1->x + p1->y - yd ; xda = p4->x + p4->y - yd ;
    di1 = pa->y + pa->x - p1->x - p1->y;
    if ((di1 <= 0) && (pb->y >= p3->y))
        dive37 (p1,p2,p3,p4,pa,yd,xia,x6,xda);
    else
    if ((di1 <= 0) && (pb->y < p3->y))
        dive49 (p1,p2,p3,p4,pb,yd,pb->y,xia,xi2,xd2,xda);
    else
    if ((di1 > 0) && (pb->y >= p3->y))
        dive33 (p1,p2,p3,p4,pa,pa->y,xi1,x6,xd1,fig,t);
    else
    if (((pa->y <= p1->y) && (pb->y >= p3->y)) || (x5 == p1->x))
        dive22 (p1,p2,p3,p4,x5,p1->y,x6,p3->y,fig,t);
}

/* funcion auxiliar a rom545 que considera al obstaculo exterior a los */
/* limites de las x de la figura */
xext545 (p1,p2,p3,p4,pa,pb,xib,xdb,xi1,xd1,xi2,xd2,yda,ds,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int xib,xdb,xi1,xd1,xi2,xd2,yda,ds,fig,t;
{
    int xia,xda,yd,di1;

    yd = (p1->y + p1->x + pa->y - pa->x) / 2;
    xia = p1->x + p1->y - yd ; xda = p4->x + p4->y - yd ;
    di1 = pa->y + pa->x - p1->x - p1->y;
    if ((ds >= 0) && (di1 <= 0))
        dive49 (p1,p2,p3,p4,pb,yd,yda,xia,xib,xdb,xda);
    else
    if ((di1 <= 0) && (ds < 0))
        dive49 (p1,p2,p3,p4,pb,yd,pb->y,xia,xi2,xd2,xda);
    else
    if ((di1 > 0) && (ds >= 0))
        dive47 (p1,p2,p3,p4,pa,pa->y,yda,xi1,xib,xdb,xd1,fig,t);
}
```

```
/* funcion que modifica ciertas coordenadas de ser necesario */
red3745 (p1,p2,p3,p4,pi,y,fg)
struct punto *p1,*p2,*p3,*p4,*pi;
int y,fg;
{
    if (((espar (p4->x + p4->y + pi->y - pi->x)) == 0) && (y < p2->y)) {
        p4->x--;
        if (fg == 3)
            p3->x--;
        else {
            p2->y--;
            p1->x++;
        }
    }
}

/* funcion que modifica ciertas coordenadas de ser necesario */
red1645 (p1,p2,p3,p4,pi,y,fg)
struct punto *p1,*p2,*p3,*p4,*pi;
int y,fg;
{
    if (((espar (p4->x + p4->y + pi->y - pi->x)) == 0) && (y < p2->y)) {
        p4->x--;
        if (fg == 1)
            p3->x--;
        else
            p2->y--;
    }
}

/* funcion que modifica ciertas coordenadas de ser necesario */
red545d (p1,p2,p3,p4,pi,y,fg)
struct punto *p1,*p2,*p3,*p4,*pi;
int y,fg;
{
    if (((espar (p4->x + p4->y + pi->y - pi->x)) == 0) && (y < p2->y)) {
        p4->x-- ; p3->x-- ; p2->x++ ; p1->x++;
    }
}

/* funcion que modifica ciertas coordenadas de ser necesario */
red545i (p1,p2,p3,p4,pi,y,fg)
struct punto *p1,*p2,*p3,*p4,*pi;
int y,fg;
{
    if (((espar (p1->x + p1->y + pi->y - pi->x)) == 0) && (y < p2->y)) {
        p4->x-- ; p3->x-- ; p2->x++ ; p1->x++;
    }
}
```

```
/* funcion que considera la presencia de obstaculos oblicuos a 135 grados */
obbli135 (cuad,seg)
struct cuadrilatero *cuad;
struct segmento *seg;
{
    struct cuadrilatero cuar;
    struct segmento segr;
    struct punto p1,p2,p3,p4,pi,pf;
    int fig,por,po,t,tr,tx,ty,cx,cy;

    po = cuad->pos ; t = seg->tipo;
    switch (po) {
    case 0 : {
        desco1 (cuad,&p1,&p2,&p3,&p4,&fig,&po);
        desco2 (seg,&pi,&pf,&t);
        switch (fig) {
        case 0 :
            rec135 (&p1,&p2,&p3,&p4,&pi,&pf,fig,t);
            break;
        case 1 : case 6 :
            if (pi.y < pf.y)
                t1t6135 (&p1,&p2,&p3,&p4,&pi,&pf,fig,t);
            else
                t1t6135 (&p1,&p2,&p3,&p4,&pf,&pi,fig,t);
            break;
        case 2 :
            if (pi.y < pf.y)
                tr2135 (&p1,&p2,&p3,&p4,&pi,&pf,fig,t);
            else
                tr2135 (&p1,&p2,&p3,&p4,&pf,&pi,fig,t);
            break;
        case 3 : case 7 :
            if (pi.y < pf.y)
                t3t7135 (&p1,&p2,&p3,&p4,&pi,&pf,fig,t);
            else
                t3t7135 (&p1,&p2,&p3,&p4,&pf,&pi,fig,t);
            break;
        case 4 :
            if (pi.y < pf.y)
                ro4135 (&p1,&p2,&p3,&p4,&pi,&pf,fig,t);
            else
                ro4135 (&p1,&p2,&p3,&p4,&pf,&pi,fig,t);
            break;
        case 5 :
            if (pi.y < pf.y)
                ro5135 (&p1,&p2,&p3,&p4,&pi,&pf,fig,t);
            else
                ro5135 (&p1,&p2,&p3,&p4,&pf,&pi,fig,t);
            break;
        }
    }
    break;
    case 1 : case 2 : case 3 : {
        constrans (cuad,&tx,&ty);
        consrot (cuad,&cx,&cy);
```

```
        rotdir (&cuar,cuad,tx,ty,cx,cy);
        rtdirs (&segr,seg,po,tx,ty,cx,cy);
        desco1 (&cuar,&p1,&p2,&p3,&p4,&fig,&por);
        desco2 (&segr,&pi,&pf,&tr);
        switch (tr) {
        case 0 : case 1 :
            break;
        case 2 :
            obbli45 (&cuar,&segr);
            break;
        case 3 :
            switch (fig) {
            case 0 :
                break;
            case 1 : case 6 :
                t1t6135 (&p1,&p2,&p3,&p4,&pf,&pi,fig,t);
                break;
            case 2 :
                tr2135 (&p1,&p2,&p3,&p4,&pf,&pi,fig,t);
                break;
            case 3 : case 7 :
                t3t7135 (&p1,&p2,&p3,&p4,&pf,&pi,fig,t);
                break;
            case 4 :
                ro4135 (&p1,&p2,&p3,&p4,&pf,&pi,fig,t);
                break;
            case 5 :
                ro5135 (&p1,&p2,&p3,&p4,&pf,&pi,fig,t);
                break;
            }
            break;
        }
        constrans (&cuar,&tx,&ty) ; consrot (&cuar,&cx,&cy);
        invierte (po,tx,ty,cx,cy);
    }
    break;
    }
}

/* funcion que divide un rectangulo por un obstaculo oblicuo a 135 grados */
rec135 (p1,p2,p3,p4,pa,pb,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int fig,t;
{
    int x5,y5,x6,y6;

        y5 = pa->y + pa->x - p3->x ; y6 = pa->y + pa->x - p1->x;
        x5 = pa->y + pa->x - p1->y ; x6 = pa->y + pa->x - p3->y ;
        if ((pa->x < p3->x) && (pa->y > p1->y) && (pb->x > p1->x) &&
            (pb->y < p3->y))
          dive46 (p1,p2,p3,p4,pa,pb,p1->x,p1->x,p3->x,p3->x,fig,t);
        else
        if ((y5 > p1->y) &&(y5 < p3->y))
            derecho (p1,p2,p3,p4,pa,pb,x6,y5,y6,fig,t);
        else
```

```
        if (y5 == p4->y)
            vertice (p1,p2,p3,p4,pa,pb,x6,y5,y6,fig,t);
        else
        if ((x5 > p1->x) && (x5 < p3->x))
          if ((y6 < p3->y) || (y6 == p3->y))
              auxil0 (p1,p2,p3,p4,pa,pb,x5,y6,p3->x,p3->x,fig,t);
          else
          if (x6 > p1->x)
              auxil3 (p1,p2,p3,p4,pa,pb,x5,p1->x,p1->x,x6,p3->x,p3->x,fig,t);
}


/* funcion que considera el obstaculo incidente por el lado derecho del */
/* rectangulo */
derecho (p1,p2,p3,p4,pa,pb,x6,y5,y6,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int x6,y5,y6,fig,t;
{
          if ((y6 < p3->y) || (y6 == p3->y))
            laderr (p1,p2,p3,p4,pa,pb,y5,y6,fig,t);
          else
          if (x6 > p1->x)
            if ((pa->x >= p3->x) && (pb->y >= p3->y))
              dive31 (p1,p2,p3,p4,pb,x6,p3->y,p3->x,y5,p1->y,fig,t);
            else
            auxil2 (p1,p2,p3,p4,pa,pb,y5,x6,fig,t);


/* funcion que considera el obstaculo incidente por el vertice inf. der. del
/* rectangulo */
vertice (p1,p2,p3,p4,pa,pb,x6,y5,y6,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int x6,y5,y6,fig,t;
{
          if ((y6 < p3->y) || (y6 == p3->y))
            laderr (p1,p2,p3,p4,pa,pb,y5,y6,fig,t);
          else
          if (x6 > p1->x)
            if ((pa->x >= p3->x) && (pb->y >= p3->y))
              dive22 (p1,p2,p3,p4,p3->x,y5,x6,p1->y,fig,t);
            else
            auxil2 (p1,p2,p3,p4,pa,pb,y5,x6,fig,t);
}

/* funcion que considera al obstaculo comprendido entre las ordenadas  */
/* maximas del rectangulo */
laderr (p1,p2,p3,p4,pa,pb,y5,y6,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int y5,y6,fig,t;
{
    if ((pa->x >= p3->x) && (pb->x <= p1->x))
        dive20 (p1,p2,p3,p4,p1->x,y6,p3->x,y5,t);
    else
    if ((pa->x >= p3->x) && (pb->x > p1->x))
        dive31 (p1,p2,p3,p4,pb,pb->x,p3->y,p3->x,y5,p1->y,fig,t);
```

```
    else
    if ((pa->x < p3->x) && (pb->x <= p1->x))
        dive30 (p1,p2,p3,p4,pa,p1->x,y6,pa->x,p3->y,p1->y,fig,t);
}

/* funcion complementaria a laderr */
auxil2 (p1,p2,p3,p4,pa,pb,y5,x6,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int y5,x6,fig,t;
{
    if ((pa->x >= p3->x) && (pb->y < p3->y))
        dive31 (p1,p2,p3,p4,pb,pb->x,p3->y,p3->x,y5,p1->y,fig,t);
    else
    if ((pa->x < p3->x) && (pb->y >= p3->y))
        dive33 (p1,p2,p3,p4,pa,pa->y,p1->x,x6,p3->x,fig,t);
}

/* funcion que divide un trapecio rectangulo izquierdo o un triangulo */
/* rectangulo izquierdo por un obstaculo oblicuo a 135 grados */
t1t6135 (p1,p2,p3,p4,pa,pb,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int fig,t;
{
    int x5,y5,x6,y6,xd,xd1;

    y5 = pa->y + pa->x - p3->x ; y6 = pa->y + pa->x - p1->x;
    x5 = pa->y + pa->x - p1->y ; x6 = pa->y + pa->x - p3->y ;
    xd = p4->x + p4->y - pa->y ; xd1 = p4->x + p4->y - pb->y;
    if ((pa->x < p4->x) && (pa->y > p1->y) && (pb->x > p1->x) &&
          (pb->y < p3->y))
          dive46 (p1,p2,p3,p4,pa,pb,p1->x,p1->x,xd1,xd,fig,t);
    else
    if ((x5 > p1->x) && (x5 < p4->x))
          if ((y6 < p2->y) || (y6 == p2->y)) {
              xd = p4->y + p4->x - y6;
              auxil0 (p1,p2,p3,p4,pa,pb,x5,y6,xd,xd1,fig,t);
          }
          else
          if ((fig == 1) && (x6 > p1->x))
              auxil3 (p1,p2,p3,p4,pa,pb,x5,p1->x,p1->x,x6,xd,xd1,fig,t);
}

/* funcion auxiliar a rec135 y a t1t6135 utilizada cuando el obstaculo */
/* incide por el lado izquierdo */
auxil0 (p1,p2,p3,p4,pa,pb,xi,yi,xd1,xd2,fg,ti)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int xi,yi,xd1,xd2,fg,ti;
{
    if ((pa->y <= p1->y) && (pb->x <= p1->x))
        ainbiz (p1,p2,p3,p4,pb,xi,xd1,yi,fg,ti);
    else
    if ((pa->y <= p1->y) && (pb->x > p1->x))
        dive32 (p1,p2,p3,p4,pb,xi,pb->y,p1->x,xd2,fg,ti);
    else
    if  ((pa->y > p1->y) && (pb->x <= p1->x))
```

```
        dive30 (p1,p2,p3,p4,pa,p1->x,yi,pa->x,p3->y,p1->y,fg,ti);
}

/* funcion auxiliar a auxil0 que supone al punto inicial del obstaculo */
/* inferior a p1 y al final a la izquierda de p1 */
pinbiz (p1,p2,p3,p4,pb,xi,xd,yi,fg,t)
struct punto *p1,*p2,*p3,*p4,*pb;
int xi,xd,yi,fg,t;
{
        if (yi < p2->y)
            dive32 (p1,p2,p3,p4,pb,xi,yi,p1->x,xd,fg,t);
        else
        if (((fg == 1) || (fg == 0)) && (yi == p3->y))
            dive22 (p1,p2,p3,p4,xi,p1->y,p1->x,p3->y,fg,t);


/* funcion que divide un trapecio rectangulo derecho por un obstaculo */
/* oblicuo a 135 grados */
pr2135 (p1,p2,p3,p4,pa,pb,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int fig,t;
{
    int ds,x5,x6,y5,y6,xi,xi1;

    y5 = pa->y + pa->x - p3->x ; y6 = pa->y + pa->x - p2->x;
    x5 = pa->y + pa->x - p1->y ; x6 = pa->y + pa->x - p3->y ;
    ds = pb->y - pb->x + p1->x - p1->y;
    xi = pa->y - p1->y + p1->x ; xi1 = pb->y - p1->y + p1->x;
    if ((pa->x < p3->x) && (pa->y > p1->y) &&
       (((ds < 0) && (pb->x < p2->x)) ||
        ((pb->y < p2->y) && (pb->x >= p2->x) && (pb->x <= p3->x))))
        dive46 (p1,p2,p3,p4,pa,pb,xi,xi1,p3->x,p3->x,fig,t);
    else
    if ((y5 > p1->y) && (y5 < p3->y))
        ladder (p1,p2,p3,p4,pa,pb,y5,fig,t);
    else
    if (y5 == p4->y)
        pinder (p1,p2,p3,p4,pa,pb,y5,fig,t);
    else
    if ((x5 > p1->x) && (x5 < p4->x))
        ladinf (p1,p2,p3,p4,pa,pb,x5,fig,t);
}

/* funcion que considera el obstaculo incidente por el lado derecho del */
/* trapecio */
ladder (p1,p2,p3,p4,pa,pb,y5,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int fig,t;
{
    int ds,x6,y6,xi,xi1,yi;

    ds = pb->y - pb->x + p1->x - p1->y;
    xi1 = (pa->x + pa->y + p1->x - p1->y) / 2;
    xi = y5 - p1->y + p1->x ; yi = xi1 - p1->x + p1->y;
    x6 = pa->y + pa->x - p3->y ; y6 = pa->y + pa->x - p2->x;
```

```
    if ((y6 < p3->y) || (y6 == p3->y))
        infsup (p1,p2,p3,p4,pa,pb,xi,xi1,yi,y5,y6,ds,fig,t);
    else
    if (x6 > p2->x)
        if ((pa->x >= p3->x) && (pb->y >= p3->y))
            dive31 (p1,p2,p3,p4,pb,x6,p3->y,p3->x,y5,p1->y,fig,t);
        else {
            xi = pa->y - p1->y + p1->x;
            auxil4 (p1,p2,p3,p4,pa,pb,xi,x6,y5,fig,t);
        }
}

/* funcio auxiliar a ladder que supone al obstaculo comprendido entre los */
/* limites inferior y superior del trapecio */
infsup (p1,p2,p3,p4,pa,pb,xi,xi1,yi,y5,y6,ds,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int xi,xi1,yi,y5,y6,ds,fig,t;
{
        if ((pa->x >= p3->x) && (ds >= 0)) {
            if (y6 < p3->y)
                dive48 (p1,p2,p3,p4,pa,y5,yi,xi,xi1,p3->x,p3->x,fig,t);
            else
            if (y6 == p3->y)
                dive33 (p1,p2,p3,p4,pa,y5,xi,p2->x,p3->x,fig,t);
        }
        else
        if ((pa->x >= p3->x) && (ds < 0)) {
            if (pb->x < p2->x) {
                yi = pb->x + p1->y - p1->x;
                dive45 (p1,p2,p3,p4,pb,pb->x,yi,p3->x,y5,p1->y,fig);
            }
            else
            if (pb->x >= p2->x)
                dive31 (p1,p2,p3,p4,pb,pb->x,p3->y,p3->x,y5,p1->y,fig,t);
        }
        else
        if ((pa->x < p3->x) && (ds >= 0)) {
              xi = pa->y - p1->y + p1->x;
              dive48 (p1,p2,p3,p4,pa,y5,yi,xi,xi1,p3->x,p3->x,fig,t);
        }
}

/* funcion que considera que el obstaculo coincide con el vertice */
/* inferior derecho */
pinder (p1,p2,p3,p4,pa,pb,y5,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int fig,t;
{
    int y6,x6,ds,yd,xi,xi1;

    ds = pb->y - pb->x + p1->x - p1->y;
    x6 = pa->y + pa->x - p3->y ; y6 = pa->y + pa->x - p2->x;
    yd = (pa->x + pa->y + p1->y - p1->x) / 2;
    xi1 = (pa->x + pa->y + p1->x - p1->y) / 2;
    xi = pa->y - p1->y + p1->x;
```

```
    if ((y6 < p3->y) || (y6 == p3->y))
        maxmin (p1,p2,p3,p4,pa,pb,xi,xi1,yd,y5,y6,ds,fig,t);
    else
    if (x6 > p2->x)
        if ((pa->x >= p3->x) && (pb->y >= p3->y))
            dive22 (p1,p2,p3,p4,p3->x,y5,x6,p3->y,fig,t);
        else
            auxil4 (p1,p2,p3,p4,pa,pb,xi,x6,y5,fig,t);
}

/* funcion auxiliar a pinder que considera al obstaculo comprendido entre */
/* las ordenadas maxima y minima del trapecio */
maxmin (p1,p2,p3,p4,pa,pb,xi,xi1,yd,y5,y6,ds,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int xi,xi1,yd,y5,y6,ds,fig,t;
{
        if ((pa->x >= p3->x) && (ds >= 0))
            aexbex (p1,p2,p3,p4,pb,p4->x,xi1,yd,y6,fig,t);
        else
        if ((pa->x >= p3->x) && (ds < 0)) {
            xi1 = pb->y - p1->y + p1->x;
            dive32 (p1,p2,p3,p4,pb,p4->x,pb->y,xi1,p3->x,fig,t);
        }
        else
        if ((pa->x < p3->x) && (ds >= 0))
            dive48 (p1,p2,p3,p4,pa,pa->y,yd,xi,xi1,p3->x,p3->x,fig,t);


/* funcion auxiliar a maxmin que considera a los extremos del obstaculo */
/* exteriores al trapecio */
aexbex (p1,p2,p3,p4,pb,xin,xi,yd,y6,fig,t)
struct punto *p1,*p2,*p3,*p4,*pb;
int xin,xi,yd,y6,fig,t;
{
    if (y6 < p3->y)
        dive35 (p1,p2,p3,p4,pb,xin,yd,xi,p3->x,fig,t);
    else
    if (y6 == p3->y)
        dive22 (p1,p2,p3,p4,xin,p4->y,p2->x,p2->y,fig,t);
}

/* funcion auxiliar a ladder y pinder */
auxil4 (p1,p2,p3,p4,pa,pb,xi,x6,y5,fg,ti)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int xi,x6,y5,fg,ti;
{
    if ((pa->x >= p3->x) && (pb->y < p3->y))
        dive31 (p1,p2,p3,p4,pb,pb->x,p3->y,p3->x,y5,p1->y,fg,ti);
    else
    if ((pa->x < p3->x) && (pb->y >= p3->y))
        dive33 (p1,p2,p3,p4,pa,pa->y,xi,x6,p3->x,fg,ti);
}

/* funcion que considera que el obstaculo es incidente por el lado inferior */
ladinf (p1,p2,p3,p4,pa,pb,x5,fig,t)
```

```
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int fig,t;
{
    int ds,x6,y6,yd,xi,xi1;

    ds = pb->y - pb->x + p1->x - p1->y;      -!
    x6 = pa->y + pa->x - p3->y ; y6 = pa->y + pa->x - p2->x;
    yd = (pa->x + pa->y + p1->y - p1->x) / 2;
    xi1 = (pa->x + pa->y + p1->x - p1->y) / 2;
    xi = pa->y - p1->y + p1->x;
    if ((y6 < p3->y) || (y6 == p3->y)) {
        if ((pa->y <= p1->y) && (ds >= 0))
            aexbex (p1,p2,p3,p4,pb,x5,xi1,yd,y6,fig,t);
        else
        if ((pa->y <= p1->y) && (ds < 0)) {
            xi1 = pb->y - p1->y + p1->x;
            dive32 (p1,p2,p3,p4,pb,x5,pb->y,xi1,p3->x,fig,t);
        }
        else
        if  ((pa->y > p1->y) && (ds >= 0))
            dive48 (p1,p2,p3,p4,pa,pa->y,yd,xi,xi1,p3->x,p3->x,fig,t);
    }
    else
    if (x6 > p2->x) {
        xi1 = pb->y - p1->y + p1->x;
        auxil3 (p1,p2,p3,p4,pa,pb,x5,xi,xi1,x6,p3->x,p3->x,fig,t);
    }
}

/* funcion que divide un trapecio isosceles o un triangulo rectangulo */
/* isosceles por un obstaculo oblicuo a 135 grados */
t3t7135 (p1,p2,p3,p4,pa,pb,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int fig,t;
{
    int ds,x5,x6,y6,xi,xi1,xd,xd1,yd;

    ds = pb->y - pb->x + p1->x - p1->y;
    xi = pa->y - p1->y + p1->x ; xi1 = pb->y - p1->y + p1->x;
    xd = p4->y + p4->x - pa->y ; xd1 = p4->y + p4->x - pb->y;
    yd = (pa->y + pa->x + p1->y - p1->x) / 2;
    y6 = pa->y + pa->x - p2->x;
    x5 = pa->y + pa->x - p1->y ; x6 = pa->y + pa->x - p2->y ;
    if ((pa->x < p4->x) && (pa->y > p1->y) &&
       (((ds < 0) && (pb->x < p2->x)) ||
        ((pb->y < p2->y) && (pb->x >= p2->x) && (pb->x <= p3->x))))
        dive46 (p1,p2,p3,p4,pa,pb,xi,xi1,xd1,xd,fig,t);
   else
   if ((x5 > p1->x) && (x5 < p4->x))
       if ((fig == 3) && (x6 > p2->x))
           auxil3 (p1,p2,p3,p4,pa,pb,x5,xi,xi1,x6,xd,xd1,fig,t);
       else
       if ((y6 < p2->y) || (y6 == p2->y)) {
           if ((pa->y <= p1->y) && (ds >= 0))
               abe37135 (p1,p2,p3,p4,pb,x5,y6,yd,fig,t);
```

```
            else
            if ((pa->y <= p1->y) && (ds < 0))
                dive32 (p1,p2,p3,p4,pb,x5,pb->y,xi1,xd1,fig,t);
            else
            if  ((pa->y > p1->y) && (ds >= 0))
                aibe3135 (p1,p2,p3,p4,pa,xi,xd,y6,yd,fig,t);
        }
}

/* funcion auxiliar a t3t7135 que considera los extremos del obstaculo */
/* exteriores a la figura */
ibe37135 (p1,p2,p3,p4,pb,x5,y6,yd,fig,t)
truct punto *p1,*p2,*p3,*p4,*pb;
int x5,y6,yd,fig,t;
{
    int xi,xd;

    xi = yd - p1->y + p1->x ; xd = p4->x + p4->y - yd;
    if (y6 < p2->y)
        dive35 (p1,p2,p3,p4,pb,x5,yd,xi,xd,fig,t);
    else
    if ((fig == 3) && (y6 == p2->y))
        dive22 (p1,p2,p3,p4,x5,p1->y,p2->x,p2->y,fig,t);
}

/* funcion auxiliar a t3t7135 que considera al extremo inicial del obstaculo
 * interior y al final exterior a la figura */
aibe3135 (p1,p2,p3,p4,pa,xi,xd,y6,yd,fig,t)
truct punto *p1,*p2,*p3,*p4,*pa;
int xi,xd,y6,yd,fig,t;
{
    int xis,xds;

    xis = yd - p1->y + p1->x ; xds = p4->x + p4->y - yd;
    if (y6 < p2->y)
        dive48 (p1,p2,p3,p4,pa,pa->y,yd,xi,xis,xds,xd,fig,t);
    else
    if ((fig == 3) && (y6 == p2->y))
        dive33 (p1,p2,p3,p4,pa,pa->y,xi,p2->x,xd,fig,t);
}

/* funcion que divide un romboide agudo por un obstaculo oblicuo a 135 grados
ro4135 (p1,p2,p3,p4,pa,pb,fig,t)
truct punto *p1,*p2,*p3,*p4,*pa,*pb;
int fig,t;
{
    int di1,ds,x5,x6,xi1,xd1,yd,xai,xad,xbi,xbd;

    yd = (pa->y + pa->x + p1->y - p1->x) / 2;
    xi1 = yd - p1->y + p1->x ; xd1 = yd - p4->y + p4->x;
    xai = pa->y - p1->y + p1->x ; xad = pa->y - p4->y + p4->x ;
    xbi = pb->y - p1->y + p1->x ; xbd = pb->y - p4->y + p4->x ;
    x5 = pa->y + pa->x - p1->y ; x6 = pa->y + pa->x - p2->y ;
    di1 = pa->y - pa->x + p4->x - p4->y;
    ds = pb->y - pb->x + p1->x - p1->y;
```

```
    if ((pa->y > p1->y) && (di1 > 0) && (ds < 0) && (pb->y < p3->y))
        dive46 (p1,p2,p3,p4,pa,pb,xai,xbi,xbd,xad,fig,t);
    else
    if (x6 >= p2->x)
        xin135 (p1,p2,p3,p4,pa,pb,x5,x6,fig,t);
    else
    if ((x6 < p2->x) && (x5 > p4->x))
        xex135 (p1,p2,p3,p4,pa,pb,xi1,xd1,yd,ds,fig,t);
    else
    if ((x5 > p1->x) && (x5 < p4->x))
        if (pa->y <= p1->y) {
            if (ds >= 0)
                dive35 (p1,p2,p3,p4,pb,x5,yd,xi1,xd1,fig,t);
            else
                dive32 (p1,p2,p3,p4,pb,x5,pb->y,xbi,xbd,fig,t);
        }
        else
        if ((pa->y > p1->y) && (ds >= 0))
            dive48 (p1,p2,p3,p4,pa,pa->y,yd,xai,xi1,xad,xd1,fig,t);
}

/* funcion auxiliar a ro4135 que supone al obstaculo comprendido entre */
/* los limites minimo y maximno de las x */
xin135 (p1,p2,p3,p4,pa,pb,x5,x6,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int x5,x6,fig,t;
{
    int xi,xd,xai,xbi,xad,xbd,yd1,di1;

    xbi = pb->y - p1->y + p1->x ; xbd = pb->y - p4->y + p4->x ;
    xai = pa->y - p1->y + p1->x ; xad = pa->y - p4->y + p4->x ;
    yd1 = (pa->y + pa->x + p4->y - p4->x) / 2;
    xi = yd1 - p1->y + p1->x ; xd = yd1 - p4->y + p4->x;
    di1 = pa->y - pa->x + p4->x - p4->y;
    if ((di1 <= 0) && (pb->y >= p3->y))
        dive36 (p1,p2,p3,p4,pb,yd1,xi,x6,xd);
    else
    if ((di1 <= 0) && (pb->y < p3->y))
        dive49a (p1,p2,p3,p4,pb,yd1,pb->y,xi,xbi,xbd,xd);
    else
    if ((di1 > 0) && (pb->y >= p3->y))
        dive33 (p1,p2,p3,p4,pa,pa->y,xai,x6,xad,fig,t);
    else
    if (((pa->y <= p1->y) && (pb->y >= p3->y)) || (x5 == p4->x))
        dive22 (p1,p2,p3,p4,x5,p1->y,x6,p3->y,fig,t);
}

/* funcion auxiliar a ro4135 que supone a los extremos del obstaculo */
/* exteriores a los limites de las x */
rex135 (p1,p2,p3,p4,pa,pb,xi1,xd1,yd,ds,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int xi1,xd1,yd,ds,fig,t;
{
    int xi,xd,xai,xbi,xad,xbd,yd1,di1;
```

```
    xbi = pb->y - p1->y + p1->x ; xbd = pb->y - p4->y + p4->x ;
    xai = pa->y - p1->y + p1->x ; xad = pa->y - p4->y + p4->x ;
    yd1 = (pa->y + pa->x + p4->y - p4->x) / 2;
    xi = yd1 - p1->y + p1->x ; xd = yd1 - p4->y + p4->x;
    di1 = pa->y - pa->x + p4->x - p4->y;
    if ((di1 <= 0) && (ds >= 0))
        dive49a (p1,p2,p3,p4,pa,yd1,yd,xi,xi1,xd1,xd);
    else
    if ((di1 <= 0) && (ds < 0))
        dive49a (p1,p2,p3,p4,pb,yd1,pb->y,xi,xbi,xbd,xd);
    else
    if ((di1 > 0) && (ds >= 0))
        dive48 (p1,p2,p3,p4,pa,pa->y,yd,xai,xi1,xd1,xad,fig,t);
}

/* funcion que divide un romboide obtuso por un obstaculo */
/* oblicuo a 135 grados */
ro5135 (p1,p2,p3,p4,pa,pb,fig,t)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int fig,t;
{
    int x5,x6,xi,xi1,xd,xd1;

    x5 = pa->y + pa->x - p1->y ; x6 = pa->y + pa->x - p2->y ;
    xi = p1->y + p1->x - pa->y ; xi1 = p1->y + p1->x - pb->y;
    xd = p4->y + p4->x - pa->y ; xd1 = p4->y + p4->x - pb->y;
    if ((pa->y > p1->y) && (pb->y < p3->y))
        dive46 (p1,p2,p3,p4,pa,pb,xi,xi1,xd1,xd,fig,t);
    else
        auxil3 (p1,p2,p3,p4,pa,pb,x5,xi,xi1,x6,xd,xd1,fig,t);
}

/* funcion auxiliar para efectuar algunas particiones utilizadas por otras */
/* funciones */
auxil3 (p1,p2,p3,p4,pa,pb,xi,xi1,xi2,xs,xd1,xd2,fg,ti)
struct punto *p1,*p2,*p3,*p4,*pa,*pb;
int xi,xi1,xi2,xs,xd1,xd2,fg,ti;
{
    if ((pa->y <= p1->y) && (pb->y >= p3->y))
        dive22 (p1,p2,p3,p4,xi,p1->y,xs,p3->y,fg,ti);
    else
    if ((pa->y <= p1->y) && (pb->y < p3->y))
        dive32 (p1,p2,p3,p4,pb,xi,pb->y,xi2,xd2,fg,ti);
    else
    if ((pa->y > p1->y) && (pb->y >= p3->y))
        dive33 (p1,p2,p3,p4,pa,pa->y,xi1,xs,xd1,fg,ti);
}

/* funcion que modifica ciertas coordenadas de ser necesario */
red37135 (p1,p2,p3,p4,pi,y,fg)
struct punto *p1,*p2,*p3,*p4,*pi;
int y,fg;
{
    if (((espar (pi->x + pi->y + p1->x - p1->y)) == 0) && (y < p2->y)) {
        p1->x++;
```

```
        if (fg == 3)
            p2->x++;
        else {
            p2->y--;
            p4->x--;
        }
    }
}

/* funcion que modifica ciertas coordenadas de ser necesario */
red2135 (p1,p2,p3,p4,pi,y,fg)
struct punto *p1,*p2,*p3,*p4,*pi;
int y,fg;
{
    if (((espar (pi->x + pi->y + p1->x - p1->y)) == 0) && (y < p2->y)) {
        p1->x++ ; p2->x++;
    }
}

/* funcion que modifica ciertas coordenadas de ser necesario */
red4135d (p1,p2,p3,p4,pi,y,fg)
struct punto *p1,*p2,*p3,*p4,*pi;
int y,fg;

    if (((espar (pi->x + pi->y + p4->x - p4->y)) == 0) && (y < p2->y)) {
        p4->x-- ; p3->x-- ; p2->x++ ; p1->x++;
    }
}

/* funcion que modifica ciertas coordenadas de ser necesario */
red4135i (p1,p2,p3,p4,pi,y,fg)
struct punto *p1,*p2,*p3,*p4,*pi;
int y,fg;
{
    if (((espar (pi->x + pi->y + p1->x - p1->y)) == 0) && (y < p2->y)) {
        p4->x-- ; p3->x-- ; p2->x++ ; p1->x++;
    }
}
```

```
extern struct cuadrilatero parte [NUMPAR];

/* divide a una region en dos subregiones (caso 0) */
dive20 (pu1,pu2,pu3,pu4,x1,y1,x2,y2,ti)
struct punto *pu1,*pu2,*pu3,*pu4;
int x1,y1,x2,y2,ti;                                    -!
{
    struct punto ax1,ax2;

    formapt (&ax1,x1,y1) ; formapt (&ax2,x2,y2);
    cdrlro (&parte[0],&ax1,pu2,pu3,&ax2);
    cdrlro (&parte[2],pu1,&ax1,&ax2,pu4);
    parte [1].ident = parte [3].ident = parte [4].ident = -1;
    if ((ti == 2) && ((ax2.x == pu3->x) && (ax2.y == pu3->y)))
        triglo (&parte[0],&ax1,pu2,&ax2);
    if ((ti == 2) && ((ax1.x == pu1->x) && (ax1.y == pu1->y)))
        triglo (&parte[2],&ax1,&ax2,pu4);
    if ((ti == 3) && ((ax1.x == pu2->x) && (ax1.y == pu2->y)))
        triglo (&parte[0],&ax1,pu3,&ax2);
    if ((ti == 3) && ((ax2.x == pu4->x) && (ax2.y == pu4->y)))
        triglo (&parte[2],pu1,&ax1,&ax2);
    if (ti == 0)
        validah (pu1,pu2,y1);


/* divide a una region en dos subregiones (caso 1) */
dive21 (pu1,pu2,pu3,pu4,x1,y1,x2,y2,ti)
struct punto *pu1,*pu2,*pu3,*pu4;
int x1,y1,x2,y2,ti;
{
    struct punto ax1,ax2;

    formapt (&ax1,x1,y1) ; formapt (&ax2,x2,y2);
    triglo (&parte[0],&ax1,pu2,&ax2);
    cdrlro (&parte[2],pu1,&ax1,&ax2,pu4);
    parte [1].ident = parte [3].ident = parte [4].ident = -1;
    if (ti == 2)
        triglo (&parte[2],&ax1,&ax2,pu4);
    if (ti == 0)
        validah (pu1,pu2,y1);
}

/* divide a una region en dos subregiones (caso 2) */
dive22 (pu1,pu2,pu3,pu4,x1,y1,x2,y2,fg,ti)
struct punto *pu1,*pu2,*pu3,*pu4;
int x1,y1,x2,y2,fg,ti;
{
    struct punto ax1,ax2;

    formapt (&ax1,x1,y1) ; formapt (&ax2,x2,y2);
    cdrlro (&parte[0],pu1,pu2,&ax2,&ax1);
    cdrlro (&parte[2],&ax1,&ax2,pu3,pu4);
    parte [1].ident = parte [3].ident = parte [4].ident = -1;
    if ((ti == 1) && ((fg == 6) || (((fg == 1) || (fg == 3) || (fg == 5)) &&
        ((ax2.x == pu3->x) && (ax2.y == pu3->y)))))
```

```
        triglo (&parte[2],&ax1,&ax2,pu4);
    if (((fg == 2) || (fg == 3) || (fg == 4)) &&
        ((ax2.x == pu2->x) && (ax2.y == pu2->y)) && (ti == 1))
        triglo (&parte[0],pu1,&ax2,&ax1);
    if (fg == 7) {
        triglo (&parte[0],pu1,&ax2,&ax1);
        triglo (&parte[2],&ax1,&ax2,pu4); }
    if ((ti == 2) && (((fg == 0) || (fg == 1) || (fg == 3) || (fg == 5)) &&
        ((ax2.x == pu3->x) && (ax2.y == pu3->y))))
        triglo (&parte[2],&ax1,&ax2,pu4);
    if ((ti == 3) && (((fg == 0) || (fg == 2) || (fg == 4)) &&
        ((ax1.x == pu4->x) && (ax1.y == pu4->y))))
        triglo (&parte[2],&ax1,&ax2,pu3);
    if ((ti == 2) && (((fg == 0) || (fg == 1) || (fg == 5)) &&
        ((ax1.x == pu1->x) && (ax1.y == pu1->y))))
        triglo (&parte[0],&ax1,pu2,&ax2);
    if ((ti == 3) && ((ax2.x == pu2->x) && (ax2.y == pu2->y)))
        triglo (&parte[0],pu1,&ax2,&ax1);
    if ((fg == 0) || (fg == 1) || (fg == 2) || (fg == 6))
        validav (pu1,pu4,x1);
    if ((ti == 2) && ((fg == 2) || (fg == 3) || (fg == 4)))
        vali45 (pu1,pu4,&ax1,&ax2);
    if ((ti == 3) && ((fg == 1) || (fg == 3) || (fg == 5)))
        vali135 (pu1,pu4,&ax1,&ax2);
}

/* funcion que divide una region en tres subregiones (caso 0) */
dive30 (pu1,pu2,pu3,pu4,pb,xi,yi,xo,ys,yin,fg,t)
struct punto *pu1,*pu2,*pu3,*pu4,*pb;
int xi,yi,xo,ys,yin,fg,t;
{
    struct punto ax1,ax2,ax3;
    int sup, inf;

    formapt (&ax1,xi,yi) ; formapt (&ax2,xo,ys) ; formapt (&ax3,xo,yin);
    if ((fg == 4) && (t == 0)) {
        sup = xo + pu2->y - yi ; inf = xo - (yi - pu1->y);
        formapt (&ax2,sup,ys) ; formapt (&ax3,inf,yin); }
    if ((fg == 5) && (t == 0)) {
        sup = xo - (pu2->y - yi) ; inf = xo + yi - pu1->y;
        formapt (&ax2,sup,ys) ; formapt (&ax3,inf,yin); }
    cdrlro (&parte[0],&ax1,pu2,&ax2,pb);
    cdrlro (&parte[1],&ax3,&ax2,pu3,pu4);
    cdrlro (&parte[2],pu1,&ax1,pb,&ax3);
    parte [3].ident = parte [4].ident = -1;
    if ((((fg == 1) || (fg == 3)) && ((ax2.x == pu3->x) && (ax2.y == pu3->y)))
        || (fg == 6))
        triglo (&parte[1],&ax3,&ax2,pu4);
    if (((fg == 2) || (fg == 3)) && ((ax2.x == pu2->x) && (ax2.y == pu2->y)))
        triglo (&parte[0],&ax1,&ax2,pb);
    if (fg == 7) {
        triglo (&parte[0],&ax1,&ax2,pb);
        triglo (&parte[1],&ax3,&ax2,pu4); }
    if ((t == 2) && (pb->y >= pu2->y)) {
        triglo (&parte[0],&ax1,pu2,&ax2);
```

```
        cdrlro (&parte[3],pu1,&ax1,&ax2,&ax3); }
    if ((t == 2) && ((ax1.x == pu1->x) && (ax1.y == pu1->y)))
        triglo (&parte[2],&ax1,pb,&ax3);
    if ((t == 3) && ((ax1.x == pu2->x) && (ax1.y == pu2->y)))
        triglo (&parte[0],pb,&ax1,&ax2);
    if ((t == 3) && (fg == 6))
        triglo (&parte[1],&ax3,&ax2,pu4);
    if (t == 0)
        validah (pu1,pu2,yi);
    if ((t == 3) && ((fg == 1) || (fg == 6)))
        vali135 (pu1,pu4,pb,&ax1);
}

/* funcion que divide una region en tres subregiones (caso 1) */
divi31 (pu1,pu2,pu3,pu4,pa,xo,ys,xd,yd,yin,fg,t)
struct punto *pu1,*pu2,*pu3,*pu4,*pa;
int xo,ys,xd,yd,yin,fg,t;
{
    struct punto ax1,ax2,ax3;
    int sup, inf;

    formapt (&ax1,xo,ys) ; formapt (&ax2,xd,yd) ;
    formapt (&ax3,xo,yin);
    if ((fg == 4) && (t == 0)) {
        sup = xo + pu2->y - yd ; inf = xo - (yd - pu1->y);
        formapt (&ax2,sup,ys) ; formapt (&ax3,inf,yin); }
    if ((fg == 5) && (t == 0)) {
        sup = xo - (pu2->y - yd) ; inf = xo + yd - pu1->y;
        formapt (&ax2,sup,ys) ; formapt (&ax3,inf,yin,1); }
    cdrlro (&parte[0],pa,&ax1,pu3,&ax2);
    cdrlro (&parte[2],&ax3,pa,&ax2,pu4);
    cdrlro (&parte[3],pu1,pu2,&ax1,&ax3);
    parte [1].ident = parte [4].ident = -1;
    if ((((fg == 1) || (fg == 3)) && ((ax1.x == pu3->x) && (ax1.y == pu3->y)))
        || (fg == 6))
        triglo (&parte[0],pa,&ax1,&ax2);
    if ((((fg == 2) || (fg == 3)) && ((ax1.x == pu2->x) && (ax1.y == pu2->y)))
        triglo (&parte[3],pu1,&ax1,&ax3);
    if (fg == 7) {
        triglo (&parte[0],pa,&ax1,&ax2);
        triglo (&parte[3],pu1,&ax1,&ax3); }
    if ((t == 2) && ((ax2.x == pu3->x) && (ax2.y == pu3->y)))
        triglo (&parte[0],pa,&ax1,&ax2);
    if ((t == 2) && (pa->y <= pu1->y)) {
        triglo (&parte[2],&ax3,&ax2,pu4);
        cdrlro (&parte[0],&ax3,&ax1,pu3,&ax2); }
    if ((t == 3) && ((ax2.x == pu4->x) && (ax2.y == pu4->y)))
        triglo (&parte[2],&ax3,pa,&ax2);
    if ((t == 3) && (pa->y >= pu3->y)) {
        triglo (&parte[0],&ax2,&ax1,pu3);
        cdrlro (&parte[2],&ax3,&ax1,&ax2,pu4); }
    if (t == 0)
        validah (pu1,pu2,yd);
    if ((t == 2) && ((fg == 2) || (fg == 3)))
        vali45 (pu1,pu4,&ax3,&ax2);
```

```
}

/* funcion que divide una region en tres subregiones (caso 2) */
dive32 (pu1,pu2,pu3,pu4,pb,xin,yo,xi,xd,fg,t)
struct punto *pu1,*pu2,*pu3,*pu4,*pb;
int xin,yo,xi,xd,fg,t;
{
    struct punto ax1,ax2,ax3;

    formapt (&ax1,xin,pu1->y) ; formapt (&ax2,xi,yo) ; formapt (&ax3,xd,yo);
    cdrlro (&parte[0],pu1,&ax2,pb,&ax1);
    cdrlro (&parte[1],&ax2,pu2,pu3,&ax3);
    cdrlro (&parte[2],&ax1,pb,&ax3,pu4);
    parte [3].ident = parte [4].ident = -1;
    if (((fg == 6) || (fg == 7)) && ((t == 1) || (t == 2) || (t == 3)))
        triglo (&parte[1],&ax2,pu2,&ax3);
    if (((fg == 2) || (fg == 4)) && ((ax1.x == pu4->x) && (ax1.y == pu4->y)))
        triglo (&parte[2],&ax1,pb,&ax3);
    if ((fg == 1) || (fg == 5) && ((ax1.x == pu1->x) && (ax1.y == pu1->y)))
        triglo (&parte[0],&ax1,&ax2,pb);
    if ((fg == 2) && (pb->x >= pu3->x)) {
        triglo (&parte[2],&ax1,&ax3,pu4);
        cdrlro (&parte[0],pu1,&ax2,&ax3,&ax1); }
    if ((t == 3) && (pb->x <= pu1->x)) {
        triglo (&parte[0],pu1,&ax2,&ax1);
        cdrlro (&parte[2],&ax1,&ax2,&ax3,pu4); }
    if (((fg == 0) || (fg == 1) || (fg == 2) || (fg == 6)) && (t == 1))
        validav (pu1,pu4,xin);
    if ((t == 2) && ((fg == 2) || (fg == 3)))
      . vali45 (pu1,pu4,&ax1,pb);
    if ((t == 3) && ((fg == 1) || (fg == 3) || (fg == 6)))
        vali135 (pu1,pu4,&ax1,pb);
}

/* funcion que divide una region en tres subregiones (caso 3) */
dive33 (pu1,pu2,pu3,pu4,pa,yo,xi,xs,xd,fg,t)
struct punto *pu1,*pu2,*pu3,*pu4,*pa;
int yo,xi,xs,xd,fg,t;
{
    struct punto ax1,ax2,ax3;

    formapt (&ax1,xi,yo) ; formapt (&ax2,xs,pu3->y) ; formapt (&ax3,xd,yo);
    cdrlro (&parte[0],&ax1,pu2,&ax2,pa);
    cdrlro (&parte[2],pa,&ax2,pu3,&ax3);
    cdrlro (&parte[3],pu1,&ax1,&ax3,pu4);
    parte [1].ident = parte [4].ident = -1;
    if ((((fg == 1) || (fg == 3) || (fg == 5)) &&
        ((ax2.x == pu3->x) && (ax2.y == pu3->y))) || (fg == 6))
        triglo (&parte[2],pa,&ax2,&ax3);
    if (((fg == 2) || (fg == 3) || (fg == 4)) &&
        ((ax2.x == pu2->x) && (ax2.y == pu2->y)))
        triglo (&parte[0],&ax1,&ax2,pa);
    if ((fg == 7) && (t == 1)) {
        triglo (&parte[0],&ax1,&ax2,pa);
        triglo (&parte[2],pa,&ax2,&ax3); }
```

```
    if ((t == 3) && (fg == 2) && (pa->x >= pu3->x) &&
        ((ax2.x == pu2->x) && (ax2.y == pu2->y))) {
        triglo (&parte[0],&ax1,&ax2,&ax3);
        triglo (&parte[2],&ax3,&ax2,pu3); }
    if (((fg == 0) || (fg == 1) || (fg == 2) || (fg == 6)) && (t == 1))
        validav (pu1,pu4,xs);
    if ((t == 2) && ((fg == 2) || (fg == 3) || (fg == 4)))
        vali45 (pu1,pu4,pa,&ax2);
    if ((t == 3) && ((fg == 1) || (fg == 3) || (fg == 5)))
        vali135 (pu1,pu4,pa,&ax2);
}

/* funcion que divide una region en tres subregiones (caso 4) */
five34 (pu1,pu2,pu3,pu4,pb,xin,yo,xi,xd,fg,t)
struct punto *pu1,*pu2,*pu3,*pu4,*pb;
int xin,yo,xi,xd,fg,t;
{
    struct punto ax1,ax2,ax3;

    formapt (&ax1,xin,pu1->y) ; formapt (&ax2,xi,yo) ; formapt (&ax3,xd,yo);
    cdrlro (&parte[0],pu1,&ax2,&ax3,&ax1);
    cdrlro (&parte[1],&ax2,pu2,pu3,&ax3);
    triglo (&parte[2],&ax1,&ax3,pu4);
    parte [3].ident = parte [4].ident = -1;
    if (((fg == 7) && ((t == 1) || (t == 2))) || ((fg == 6) && (t == 2)))
        triglo (&parte[1],&ax2,pu2,&ax3);
    if (((fg == 1) || (fg == 6)) && (t == 2) &&
        ((ax1.x == pu1->x) && (ax1.y == pu1->y)))
        triglo (&parte[0],&ax1,&ax2,&ax3);
    if ((fg == 5) && ((ax1.x == pu1->x) && (ax1.y == pu1->y)))
        triglo (&parte[0],&ax1,&ax2,&ax3);
    if ((fg == 1) && (t == 1))
        validav (pu1,pu4,xin);
    if ((t == 2) && ((fg == 3) || (fg == 7)))
        vali45 (pu1,pu4,&ax1,&ax3);
}

/* funcion que divide una region en tres subregiones (caso 5) */
five35 (pu1,pu2,pu3,pu4,pb,xin,yo,xi,xd,fg,t)
struct punto *pu1,*pu2,*pu3,*pu4,*pb;
int xin,yo,xi,xd,fg,t;
{
    struct punto ax1,ax2,ax3;

    formapt (&ax1,xin,pu1->y) ; formapt (&ax2,xi,yo) ; formapt (&ax3,xd,yo);
    triglo (&parte[0],pu1,&ax2,&ax1);
    cdrlro (&parte[1],&ax2,pu2,pu3,&ax3);
    cdrlro (&parte[2],&ax1,&ax2,&ax3,pu4);
    parte [3].ident = parte [4].ident = -1;
    if ((fg == 7) && ((t == 1) || (t == 3)))
        triglo (&parte[1],&ax2,pu2,&ax3);
    if (((fg == 2) || (fg == 4)) && ((ax1.x == pu4->x) && (ax1.y == pu4->y)))
        triglo (&parte[2],&ax1,&ax2,&ax3);
    if ((fg == 2) && (t == 1))
        validav (pu1,pu4,xin);
```

```
    if (((fg == 3) || (fg == 7)) && (t == 3))
        vali135 (pu1,pu4,&ax1,&ax2);


/* funcion que divide una region en tres subregiones (caso 6) */
live36 (pu1,pu2,pu3,pu4,pa,yo,xi,xs,xd)
truct punto *pu1,*pu2,*pu3,*pu4,*pa;
nt yo,xi,xs,xd;
{
    struct punto ax1,ax2,ax3;

    formapt (&ax1,xi,yo) ; formapt (&ax2,xs,pu3->y) ; formapt (&ax3,xd,yo);
    cdrlro (&parte[0],&ax1,pu2,&ax2,&ax3);
    triglo (&parte[2],&ax3,&ax2,pu3);
    cdrlro (&parte[3],pu1,&ax1,&ax3,pu4);
    parte [1].ident = parte [4].ident = -1;
    if ((ax2.x == pu2->x) && (ax2.y == pu2->y))
        triglo (&parte[0],&ax1,&ax2,&ax3);
}

/* funcion que divide una region en tres subregiones (caso 7) */
live37 (pu1,pu2,pu3,pu4,pa,yo,xi,xs,xd)
truct punto *pu1,*pu2,*pu3,*pu4,*pa;
nt yo,xi,xs,xd;
{
    struct punto ax1,ax2,ax3;

    formapt (&ax1,xi,yo) ; formapt (&ax2,xs,pu3->y) ; formapt (&ax3,xd,yo);
    triglo (&parte[0],&ax1,pu2,&ax2);
    cdrlro (&parte[2],&ax1,&ax2,pu3,&ax3);
    cdrlro (&parte[3],pu1,&ax1,&ax3,pu4);
    parte [1].ident = parte [4].ident = -1;
    if ((ax2.x == pu3->x) && (ax2.y == pu3->y))
        triglo (&parte[2],&ax1,&ax2,&ax3);
}

/* funcion que divide una region en tres subregiones (caso 8) */
live38 (pu1,pu2,pu4,xi,yo,xs,ys,xd)
truct punto *pu1,*pu2,*pu4;
nt yo,xi,xs,ys,xd;
{
    struct punto ax1,ax2,ax3;

    formapt (&ax1,xi,yo) ; formapt (&ax2,xs,ys) ; formapt (&ax3,xd,yo);
    triglo (&parte[0],&ax1,pu2,&ax2);
    triglo (&parte[2],&ax1,&ax2,&ax3);
    cdrlro (&parte[3],pu1,&ax1,&ax3,pu4);
    parte [1].ident = parte [4].ident = -1;
}

/* funcion que divide una region en cuatro subregiones (caso 0) */
live40 (pu1,pu2,pu3,pu4,pa,pb,y1,y2,y3,y4,fg,t)
truct punto *pu1,*pu2,*pu3,*pu4,*pa,*pb;
nt y1,y2,y3,y4,fg,t;
{
```

```
   struct punto ax1,ax2,ax3,ax4;
   int su1,su2,in1,in2;

   formapt (&ax1,pa->x,y1) ; formapt (&ax2,pb->x,y2);
   formapt (&ax3,pb->x,y3) ; formapt (&ax4,pa->x,y4);
   if ((fg == 4) && (t == 0)) {
       su1 = pa->x + pu2->y - pa->y ; su2 = pb->x + pu2->y - pb->y;
       in1 = pa->x - (pa->y - pu1->y) ; in2 = pb->x - (pb->y - pu1->y);
       formapt (&ax1,su1,y1) ; formapt (&ax2,su2,y2);
       formapt (&ax3,in2,y3) ; formapt (&ax4,in1,y4); }
   if ((fg == 5) && (t == 0)) {
       su1 = pa->x - (pu2->y - pa->y) ; su2 = pb->x - (pu2->y - pb->y);
       in1 = pa->x + pa->y - pu1->y ; in2 = pb->x + pb->y - pu1->y;
       formapt (&ax1,su1,y1) ; formapt (&ax2,su2,y2);
       formapt (&ax3,in2,y3) ; formapt (&ax4,in1,y4); }
   cdrlro (&parte[0],pa,&ax1,&ax2,pb);
   cdrlro (&parte[1],&ax3,&ax2,pu3,pu4);
   cdrlro (&parte[2],&ax4,pa,pb,&ax3);
   cdrlro (&parte[3],pu1,pu2,&ax1,&ax4); parte [4].ident = -1;
   if (((fg == 1) || (fg == 3)) &&
       ((ax2.x == pu3->x) && (ax2.y == pu3->y)))
       triglo (&parte[1],&ax3,&ax2,pu4);
   if (((fg == 2) || (fg == 3)) && ((ax1.x == pu2->x) && (ax1.y == pu2->y)))
       triglo (&parte[3],pu1,&ax1,&ax4);
   validah (pu1,pu2,pa->y);


* funcion que divide una region en cuatro subregiones (caso 1) */
ive41 (pu1,pu2,pu3,pu4,pb,xi,yi,xo,ys,yin,fg)
truct punto *pu1,*pu2,*pu3,*pu4,*pb;
nt xi,yi,xo,ys,yin,fg;

   struct punto ax1,ax2,ax3,ax4;
   int xi1;

   if (fg == 1)
       xi1 = pu1->x;
   else
       xi1 = ys - pu1->y + pu1->x;
   formapt (&ax1,xi,yi) ; formapt (&ax2,xi1,ys);
   formapt (&ax3,pb->x,ys) ; formapt (&ax4,pb->x,yin);
   cdrlro (&parte[0],&ax2,pu2,pu3,&ax3);
   cdrlro (&parte[1],&ax1,&ax2,&ax3,pb);
   cdrlro (&parte[2],pu1,&ax1,pb,&ax4);
   triglo (&parte[3],&ax4,&ax3,pu4); parte [4].ident = -1;
   if (fg == 7)
       triglo (&parte[0],&ax2,pu2,&ax3);
   validah (pu1,pu2,yi);


* funcion que divide una region en cuatro subregiones (caso 2) */
ive42 (pu1,pu2,pu3,pu4,pa,xo,ys,xd,yd,yin,fg)
truct punto *pu1,*pu2,*pu3,*pu4,*pa;
nt xo,ys,xd,yd,yin,fg;
```

```
    struct punto ax1,ax2,ax3,ax4;
    int xi1;

    if (fg == 1)
        xi1 = pu1->x;
    else
        xi1 = ys - pu1->y + pu1->x;
    formapt (&ax1,xi1,ys) ; formapt (&ax2,pa->x,ys);
    formapt (&ax3,pa->x,yin) ; formapt (&ax4,xd,yd);
    cdrlro (&parte[0],&ax1,pu2,pu3,&ax2);
    triglo (&parte[1],pa,&ax2,&ax4);
    cdrlro (&parte[2],&ax3,pa,&ax4,pu4);
    cdrlro (&parte[3],pu1,&ax1,&ax2,&ax3); parte [4].ident = -1;
    if (fg == 7)
        triglo (&parte[0],&ax1,pu2,&ax2);
    validah (pu1,pu2,pa->y);
}

/* funcion que divide una region en cuatro subregiones (caso 3) */
dive43 (pu1,pu2,pu3,pu4,pa,pb,y1,y2,y3,y4)
struct punto *pu1,*pu2,*pu3,*pu4,*pa,*pb;
int y1,y2,y3,y4;
{
    struct punto ax1,ax2,ax3,ax4;

    formapt (&ax1,pa->x,y1) ; formapt (&ax2,pb->x,y2);
    formapt (&ax3,pb->x,y3) ; formapt (&ax4,pa->x,y4);
    cdrlro (&parte[0],pa,&ax1,&ax2,pb);
    triglo (&parte[1],&ax3,&ax2,pu4);
    cdrlro (&parte[2],&ax4,pa,pb,&ax3);
    cdrlro (&parte[3],pu1,pu2,&ax1,&ax4); parte [4].ident = -1;
    validah (pu1,pu2,pa->y);
}

/* funcion que divide una region en cuatro subregiones (caso 4) */
dive44 (pu1,pu2,pu3,pu4,pb,xi,yi,xo,ys,yin,fg)
struct punto *pu1,*pu2,*pu3,*pu4,*pb;
int xi,yi,xo,ys,yin,fg;
{
    struct punto ax1,ax2,ax3,ax4;
    int xd;

    if (fg == 2)
        xd = pu3->x;
    else
        xd = pu4->x + pu4->y - ys;
    formapt (&ax1,xi,yi) ; formapt (&ax2,xo,ys) ;
    formapt (&ax3,xo,yin); formapt (&ax4,xd,ys);
    cdrlro (&parte[0],&ax2,pu2,pu3,&ax4);
    triglo (&parte[1],&ax1,&ax2,&ax3);
    cdrlro (&parte[2],pu1,&ax1,pb,&ax3);
    cdrlro (&parte[3],&ax3,&ax2,&ax4,pu4); parte [4].ident = -1;
    if (fg == 7)
        triglo (&parte[0],&ax2,pu2,&ax4);
    validah (pu1,pu2,yi);
```

```
}

/* funcion que divide una region en cuatro subregiones (caso 5) */
dive45 (pu1,pu2,pu3,pu4,pa,xo,ys,xd,yd,yin,fg)
struct punto *pu1,*pu2,*pu3,*pu4,*pa;
int xo,ys,xd,yd,yin,fg;
{
    struct punto ax1,ax2,ax3,ax4;
    int xd1;

    if (fg == 1)
        xd1 = pu3->x;
    else
        xd1 = pu4->y + pu4->x - ys;
    formapt (&ax1,pa->x,ys) ; formapt (&ax2,xd1,ys);
    formapt (&ax3,xd,yd) ; formapt (&ax4,pa->x,yin);
    cdrlro (&parte[0],&ax1,pu2,pu3,&ax2);
    cdrlro (&parte[1],pa,&ax1,&ax2,&ax3);
    cdrlro (&parte[2],&ax4,pa,&ax3,pu4);
    triglo (&parte[3],pu1,&ax1,&ax4); parte [4].ident = -1;
    if (fg == 7)
        triglo (&parte[0],&ax1,pu2,&ax2);
    validah (pu1,pu2,pa->y);
}

/* funcion que divide una region en cuatro subregiones (caso 6) */
dive46 (pu1,pu2,pu3,pu4,pa,pb,x1,x2,x3,x4,fg,t)
struct punto *pu1,*pu2,*pu3,*pu4,*pa,*pb;
int x1,x2,x3,x4,fg,t;
{
    struct punto ax1,ax2,ax3,ax4;

    formapt (&ax1,x1,pa->y) ; formapt (&ax2,x2,pb->y);
    formapt (&ax3,x3,pb->y) ; formapt (&ax4,x4,pa->y);
    cdrlro (&parte[0],&ax1,&ax2,pb,pa);
    cdrlro (&parte[1],&ax2,pu2,pu3,&ax3);
    cdrlro (&parte[2],pa,pb,&ax3,&ax4);
    cdrlro (&parte[3],pu1,&ax1,&ax4,pu4); parte [4].ident = -1;
    if ((fg == 6) || (fg == 7))
        triglo (&parte[1],&ax2,pu2,&ax3);
    if (((fg == 0) || (fg == 1) || (fg ==2) || (fg == 6)) && (t == 1))
        validav (pu1,pu4,pa->x);
    if (((fg == 2) || (fg == 3) || (fg == 4) || (fg == 7)) && (t == 2))
        vali45 (pu1,pu4,pa,pb);
}

/* funcion que divide una region en cuatro subregiones (caso 7) */
dive47 (pu1,pu2,pu3,pu4,pa,yi,ys,x1,x2,x3,x4,fg,t)
struct punto *pu1,*pu2,*pu3,*pu4,*pa;
int yi,ys,x1,x2,x3,x4,fg,t;
{
    struct punto ax1,ax2,ax3,ax4;

    formapt (&ax1,x1,yi) ; formapt (&ax2,x2,ys);
    formapt (&ax3,x3,ys) ; formapt (&ax4,x4,yi);
```

```
   cdrlro (&parte[0],&ax1;&ax2,&ax3,pa);
   cdrlro (&parte[1],&ax2,pu2,pu3,&ax3);
   triglo (&parte[2],pa,&ax3,&ax4);
   cdrlro (&parte[3],pu1,&ax1,&ax4,pu4); parte [4].ident = -1;
   if ((fg == 1) && (pa->x <= pu1->x) && (t == 2)) {
       triglo (&parte[0],&ax1,&ax2,&ax3);
       triglo (&parte[2],&ax1,&ax3,&ax4); }
   if ((fg == 6) || (fg == 7))
       triglo (&parte[1],&ax2,pu2,&ax3);
   if ((fg == 1) && (t == 1))
       validav (pu1,pu4,pa->x);
   if (((fg == 3) && (fg == 7)) && (t == 2))
       vali45 (pu1,pu4,pa,&ax3);


* funcion que divide una region en cuatro subregiones (caso 8) */
ive48 (pu1,pu2,pu3,pu4,pa,yi,ys,x1,x2,x3,x4,fg,t)
truct punto *pu1,*pu2,*pu3,*pu4,*pa;
nt yi,ys,x1,x2,x3,x4,fg,t;

   struct punto ax1,ax2,ax3,ax4;

   formapt (&ax1,x1,yi) ; formapt (&ax2,x2,ys);
   formapt (&ax3,x3,ys) ; formapt (&ax4,x4,yi);
   triglo (&parte[0],&ax1,&ax2,pa);
   cdrlro (&parte[1],&ax2,pu2,pu3,&ax3);
   cdrlro (&parte[2],pa,&ax2,&ax3,&ax4);
   cdrlro (&parte[3],pu1,&ax1,&ax4,pu4); parte [4].ident = -1;
   if ((fg == 2) && (t == 3) && (pa->x >= pu3->x)) {
       triglo (&parte[0],&ax1,&ax2,&ax4);
       triglo (&parte[2],&ax4,&ax2,&ax3); }
   if (fg == 7)
       triglo (&parte[1],&ax2,pu2,&ax3);
   if ((fg == 2) && (t == 1))
       validav (pu1,pu4,pa->x);
   if (((fg == 3) || (fg == 7)) && (t == 3))
       vali135 (pu1,pu4,&ax1,&ax2);


* funcion que divide una region en cuatro subregiones (caso 9) */
ive49 (pu1,pu2,pu3,pu4,pb,yi,ys,x1,x2,x3,x4)
truct punto *pu1,*pu2,*pu3,*pu4,*pb;
nt yi,ys,x1,x2,x3,x4;

   struct punto ax1,ax2,ax3,ax4;
   int ds;

   ds = pb->y + pb->x - pu4->y - pu4->x;
   formapt (&ax1,x1,yi) ; formapt (&ax2,x2,ys);
   formapt (&ax3,x3,ys) ; formapt (&ax4,x4,yi);
   triglo (&parte[0],&ax1,&ax2,pb);
   cdrlro (&parte[1],&ax2,pu2,pu3,&ax3);
   cdrlro (&parte[2],&ax1,pb,&ax3,&ax4);
   cdrlro (&parte[3],pu1,&ax1,&ax4,pu4); parte [4].ident = -1;
   if (ds >= 0) {
```

```
        triglo (&parte[0],&ax1,&ax2,&ax3);
        triglo (&parte[2],&ax1,&ax3.&ax4); }


* funcion que divide una region en cuatro subregiones (caso 9a) */
ive49a (pu1,pu2,pu3,pu4,pb,yi,ys,x1,x2,x3,x4)
truct punto *pu1,*pu2,*pu3,*pu4.*pb;
nt yi,ys,x1,x2,x3,x4;

    struct punto ax1,ax2,ax3,ax4;
    int ds;

    ds = pb->y - pb->x + pu1->x - pu1->y ;
    formapt (&ax1,x1,yi) ; formapt (&ax2,x2,ys);
    formapt (&ax3,x3,ys) ; formapt (&ax4,x4,yi);
    cdrlro (&parte[0],&ax1,&ax2,pb,&ax4);
    cdrlro (&parte[1],&ax2,pu2,pu3,&ax3);
    triglo (&parte[2],&ax4,pb,&ax3);
    cdrlro (&parte[3],pu1,&ax2,&ax4,pu4); parte [4].ident = -1;
    if (ds >= 0) {
        triglo (&parte[0],&ax1,&ax2,&ax4);
        triglo (&parte[2],&ax4,&ax2,&ax3); }
}

/* funcion que divide una region en cinco subregiones (caso 0) */
ive50 (pu1,pu2,pu3,pu4,pa,pb,y1,y2,y3,y4,fg)
truct punto *pu1,*pu2,*pu3.*pu4.*pa,*pb;
nt y1,y2,y3,y4,fg;

    struct punto ax1,ax2,ax3,ax4,ax5;

    formapt (&ax1,pa->x,y1) ; formapt (&ax2,pb->x,y2);
    formapt (&ax3,pb->x,y3) ; formapt (&ax4,pa->x,y4);formapt (&ax5,pa->x,y2)
    cdrlro (&parte[0],&ax5,&ax4,pu3,&ax2);
    cdrlro (&parte[1],pa,&ax5.&ax2,pb);
    cdrlro (&parte[2],&ax4,pa,pb,&ax3);
    triglo (&parte[3],&ax3.&ax2,pu4);
    cdrlro (&parte[4],pu1,pu2,&ax1,&ax4);
    if ((fg == 3) && ((ax1.x == pu2->x) && (ax1.y == pu2->y)))
        triglo (&parte[4],pu1,&ax1,&ax4);
    validah (pu1,pu2,pa->y);


* funcion que divide una region en cinco subregiones (caso 1) */
ive51 (pu1,pu2,pu3,pu4,pa,pb,y1,y2,y3,y4,fg)
truct punto *pu1,*pu2,*pu3,*pu4,*pa,*pb;
nt y1,y2,y3,y4,fg;

    struct punto ax1,ax2,ax3,ax4,ax5;

    formapt (&ax1,pa->x,y1) ; formapt (&ax2,pb->x,y2);
    formapt (&ax3,pb->x,y3) ; formapt (&ax4,pa->x,y4);formapt (&ax5,pb->x,y2)
    cdrlro (&parte[0],&ax1,pu2,&ax2,&ax5);
    cdrlro (&parte[1],pa,&ax1,&ax5,pb);
    cdrlro (&parte[2],&ax4,pa,pb,&ax3);
```

```
    cdrlro (&parte[3],&ax3,&ax2,pu3,pu4);
    triglo (&parte[4],pu1,&ax1,&ax4);
    if ((fg == 3) && ((ax2.x == pu3->x) && (ax2.y == pu3->y)))
        triglo (&parte[3],&ax3,&ax2,pu4);
    validah (pu1,pu2,pa->y);
}

/* funcion que divide una region en cinco subregiones (caso 2) */
dive52 (pu1,pu2,pu3,pu4,pa,pb,y1,y2,y3,y4,fg)
struct punto *pu1,*pu2,*pu3,*pu4,*pa,*pb;
int y1,y2,y3,y4,fg;
{
    struct punto ax1,ax2,ax3,ax4,ax5;
    int x5;

    formapt (&ax1,pa->x,y1) ; formapt (&ax2,pb->x,y2);
    formapt (&ax3,pb->x,y3) ; formapt (&ax4,pa->x,y4);
    if (y1 > y2) {
        x5 = y2 - pu1->y + pu1->x;
        formapt (&ax5,x5,y2) ; formapt (&ax1,pa->x,y2);
        cdrlro (&parte[0],&ax5,pu2,pu3,&ax2);
        if (fg == 7)
            triglo (&parte[0],&ax5,pu2,&ax2);
        triglo (&parte[3],&ax3,&ax2,pu4);
        cdrlro (&parte[4],pu1,&ax5,&ax1,&ax4); }
    else
    if (y2 > y1) {
        x5 = pu4->x + pu4->y - y1;
        formapt (&ax5,x5,y1) ; formapt (&ax2,pb->x,y1);
        cdrlro (&parte[0],&ax1,pu2,pu3,&ax5);
        if (fg == 7)
            triglo (&parte[0],&ax1,pu2,&ax5);
        cdrlro (&parte[3],&ax3,&ax2,&ax5,pu4);
        triglo (&parte[4],pu1,&ax1,&ax4); }
    else
    if (y1 == y2) {
        cdrlro (&parte[0],&ax1,pu2,pu3,&ax2);
        if (fg == 7)
            triglo (&parte[0],&ax1,pu2,&ax2);
        triglo (&parte[3],&ax3,&ax2,pu4);
        triglo (&parte[4],pu1,&ax1,&ax4); }
    cdrlro (&parte[1],pa,&ax1,&ax2,pb);
    cdrlro (&parte[2],&ax4,pa,pb,&ax3);
    validah (pu1,pu2,pa->y);


  funcion que valida las subregiones generadas por un obstaculo horizontal */
alidah (pt1,pt2,ord)
truct punto *pt1,*pt2;
nt ord;

    if (ord == pt2->y)
        parte [0].ident = -1;
    else
    if (ord == pt1->y)
```

```
        parte [2].ident = -1;
}

* funcion que valida las subregiones generadas por un obstaculo vertical */
alidav (pt1,pt2,ab)
truct punto *pt1,*pt2;
nt ab;

   if (ab == pt1->x)
       parte [0].ident = -1;
   else
   if (ab == pt2->x)
       parte [2].ident = -1;


* funcion que valida las subregiones generadas por un obstaculo oblicuo 45 *
ali45 (pt1,pt2,pa1,pa2)
truct punto *pt1,*pt2,*pa1,*pa2;

   int d1,d2;

   d1 = pa1->y - pa1->x + pt1->x - pt1->y;
   d2 = pa2->y - pa2->x + pt1->x - pt1->y;
   if ((d1 == 0) && (d2 == 0))
       parte [0].ident = -1;
   d1 = pa1->y - pa1->x + pt2->x - pt2->y;
   d2 = pa2->y - pa2->x + pt2->x - pt2->y;
   if ((d1 == 0) && (d2 == 0))
       parte [2].ident = -1;


* funcion que valida las subregiones generadas por un obstaculo oblicuo 135
ali135 (pt1,pt2,pa1,pa2)
truct punto *pt1,*pt2,*pa1,*pa2;

   int d1,d2;

   d1 = pa1->y + pa1->x - pt1->x - pt1->y;
   d2 = pa2->y + pa2->x - pt1->x - pt1->y;
   if ((d1 == 0) && (d2 == 0))
       parte [0].ident = -1;
   d1 = pa1->y + pa1->x - pt2->x - pt2->y;
   d2 = pa2->y + pa2->x - pt2->x - pt2->y;
   if ((d1 == 0) && (d2 == 0))
       parte [2].ident = -1;


* funcion que rota las particiones a la posicion original de la figura */
* que se dividio */
vierte (pst,tr1,tr2,ro1,ro2)
t pst,tr1,tr2,ro1,ro2;

   struct cuadrilatero  cdorig;
   int i;
```

```
    for (i = 0; i < NUMPAR; i++)
        if (parte [i].ident != -1) {
            rotinv (&cdorig,&parte[i],pst,tr1,tr2,ro1,ro2);
            obveci (&parte[i],&cdorig);
        }


/* funcion que forma la estructura de un cuadrilatero */
drlro (r,pii,pis,pds,pdi)
truct cuadrilatero *r;
truct punto *pii,*pis,*pds,*pdi;
{
    int i,fig,pst;

    figpos (pii,pis,pds,pdi,&fig,&pst);
    r->infizq.x = pii->x ; r->infizq.y = pii->y;
    r->supizq.x = pis->x ; r->supizq.y = pis->y;
    r->supder.x = pds->x ; r->supder.y = pds->y;
    r->infder.x = pdi->x ; r->infder.y = pdi->y;
    for (i = 0; i < NUMVER; i++)
        r->indob[i] = 0;
    r->figura = fig;
    r->pos = pst;
    r->ident = 0;


/* funcion que forma la estructura de un triangulo */
riglo (r,piz,ps,pd)
truct cuadrilatero *r;
truct punto *piz,*ps,*pd;

    int i,fig,pst;

    figpos (piz,ps,NULL,pd,&fig,&pst);
    r->infizq.x = piz->x ; r->infizq.y = piz->y;
    r->supizq.x = ps->x ; r->supizq.y = ps->y;
    r->infder.x = pd->x ; r->infder.y = pd->y;
    for (i = 0; i < NUMVER ; i++)
        r->indob[i] = 0;
    r->figura = fig;
    r->pos = pst;
    r->ident = 0;


/* funcion que obtiene el tipo de figura y su posicion */
igpos (p1,p2,p3,p4,fgr,psn)
truct punto *p1, *p2, *p3, *p4;
nt *fgr,*psn;

    int t1,t2,t3,t4;

 if (p3 != NULL) {
* es cuadrilatero */
    obtipo (p1,p2,&t1);
    obtipo (p2,p3,&t2);
```

```
    obtipo (p4,p3,&t3);
    obtipo (p1,p4,&t4);
/* es rectangulo */
    if ((t2 == 0) && (t4 == 0)) {
        *psn = 0;
        if ((t1 == 1) && (t3 == 1))
            *fgr = 0;
        else
/* es trapecio rectangulo izquierdo */
        if ((t1 == 1) && (t3 == 3))
            *fgr = 1;
        else
        if ((t1 == 3) && (t3 == 1)) {
            *fgr = 1 ; *psn = 2; }
/* es trapecio rectangulo derecho */
        else
        if ((t1 == 2) && (t3 == 1))
            *fgr = 2;
        else
        if ((t1 == 1) && (t3 == 2)) {
            *fgr = 2 ; *psn = 2; }
/* es trapecio isosceles */
        else
        if ((t1 == 2) && (t3 == 3))
            *fgr = 3;
        else
        if ((t1 == 3) && (t3 == 2)) {
            *fgr = 3 ; *psn = 2; }
/* es romboide agudo */
        else
        if ((t1 == 2) && (t3 == 2))
            *fgr = 4;
/* es romboide obtuso */
        else
            *fgr = 5;
    }
    else
    if ((t1 == 1) && (t3 == 1)) {
        *psn = 1;
/* es trapecio rectangulo izquierdo */
        if ((t2 == 0) && (t4 == 2))
            *fgr = 1;
        else
        if ((t2 == 2) && (t4 == 0))  {
            *fgr = 1 ; *psn = 3; }
/* es trapecio rectangulo derecho */
        else
        if ((t2 == 3) && (t4 == 0))
            *fgr = 2;
        else
        if ((t2 == 0) && (t4 == 3)) {
            *fgr = 2 ; *psn = 3; }
        else
/* es trapecio isoceles */
        if ((t2 == 3) && (t4 == 2))
```

```
            *fgr = 3;
        else
        if ((t2 == 2) && (t4 == 3))  {
            *fgr = 3 ; *psn = 3; }
/* es romboide agudo */
        else
        if ((t2 == 3) && (t4 == 3))
            *fgr = 4;
/* es romboide obtuso */
        else
            *fgr = 5;
    }
  }
  else {
/* es triangulo */
    obtipo (p1,p2,&t1);
    obtipo (p4,p2,&t2);
    obtipo (p1,p4,&t3);
/* es triangulo rectangulo izquierdo */
    if ((t1 == 1) && (t2 == 3) && (t3 == 0)) {
        *fgr = 6 ; *psn = 0; }
    else
    if ((t1 == 1) && (t2 == 0) && (t3 == 2)) {
        *fgr = 6 ; *psn = 1; }
    else
    if ((t1 == 3) && (t2 == 0) && (t3 == 1)) {
        *fgr = 6 ; *psn = 2; }
    else
    if ((t1 == 2) && (t2 == 1) && (t3 == 0)) {
        *fgr = 6 ; *psn = 3; }
    else
/* es triangulo rectangulo isosceles */
    if ((t1 == 2) && (t2 == 3) && (t3 == 0)) {
        *fgr = 7 ; *psn = 0; }
    else
    if ((t1 == 1) && (t2 == 3) && (t3 == 2)) {
        *fgr = 7 ; *psn = 1; }
    else
    if ((t1 == 3) && (t2 == 0) && (t3 == 2)) {
        *fgr = 7 ; *psn = 2; }
    else
    if ((t1 == 3) && (t2 == 2) && (t3 == 1)) {
        *fgr = 7 ; *psn = 3; }
  }
}

/* funcion que obtiene un punto dado por sus coordenadas */
formapt (pt,ab,ord)
struct punto *pt;
int ab,ord;
{
   pt->x = ab;
   pt->y = ord;
```

```
/* funcion que descompone una estructura de segmento en sus partes.*/
desco2 (seg,p1,p2,ti)
struct segmento *seg;
struct punto *p1,*p2;
int *ti;
{
    p1->x = seg->inicial.x ; p1->y = seg->inicial.y;
    p2->x = seg->final.x ; p2->y = seg->final.y;
    *ti = seg->tipo;
}
```

```
xtern int ndn;

/* funcion que lee los datos de la grafica del archivo y forma la */
/* lista ligada de figuras */
truct lstcdr *leegra (lsf,ar)
truct lstcdr *lsf;                              -!
ILE *ar;
{
    struct elemento e;
    struct cuadrilatero cdr;
    int i,c,ind,idt,fig,pst,num = 0;

    while ((c = fscanf (ar,"%5d%5d%5d%5d",&ind,&idt,&fig,&pst)) != EOF)
        if (ind == 1) {
            switch (num) {
            case 0 :
                break;
            default :
                lsf = listaf (lsf,&e);
                break;
            }
            e.basico.ident = idt ; e.basico.figura = fig ; e.basico.pos = pst
            for (i = 0; i < NUMVER; i++)
                fscanf (ar,"%5d",&e.basico.indob[i]);
            fscanf (ar,"%5d%5d",&e.basico.infizq.x,&e.basico.infizq.y);
            fscanf (ar,"%5d%5d",&e.basico.supizq.x,&e.basico.supizq.y);
            switch (e.basico.figura) {
            case 0 : case 1 : case 2 : case 3 : case 4 : case 5 :
                fscanf (ar,"%5d%5d",&e.basico.supder.x,&e.basico.supder.y);
                break;
            case 6 : case 7 :
                break;
            }
            fscanf (ar,"%5d%5d\n",&e.basico.infder.x,&e.basico.infder.y);
            e.vecino = NULL ; num++;
        }
        else
        if (ind == 0) {
            cdr.ident = idt ; cdr.figura = fig ; cdr.pos = pst;
            for (i = 0; i < NUMVER; i++)
                fscanf (ar,"%5d",&cdr.indob[i]);
            fscanf (ar,"%5d%5d",&cdr.infizq.x,&cdr.infizq.y);
            fscanf (ar,"%5d%5d",&cdr.supizq.x,&cdr.supizq.y);
            switch (cdr.figura) {
            case 0 : case 1 : case 2 : case 3 : case 4 : case 5 :
                fscanf (ar,"%5d%5d",&cdr.supder.x,&cdr.supder.y);
                break;
            case 6 : case 7 :
                break;
            }
            fscanf (ar,"%5d%5d\n",&cdr.infder.x,&cdr.infder.y);
            e.vecino = lstrec (e.vecino,&cdr);
        }
    lsf = listaf (lsf,&e);
    ndn = num;
```

```
   return (lsf);


  funcion que efectua la lista ligada de figuras a partir de los */
  datos leidos del archivo */
truct lstcdr *listaf (ls,elem)
truct lstcdr *ls;
truct elemento *elem;

   struct lstcdr *q, *p;
   char *malloc();

   q = (struct lstcdr *) malloc(sizeof(struct lstcdr));
   copelem (&q->nodo,elem);
   q->next = NULL;
   if (ls == NULL)
       ls = q;
   else {
       p = ls;
       while (p->next != NULL)
           p = p->next;
       p->next = q;
   }
   return (ls);


* funcion que obtiene el campo de informacion de la lista ligada */
* general de figuras */
opelem (cuad,cdro)
truct elemento *cuad, *cdro;

   int i;

   cuad->basico.infizq.x = cdro->basico.infizq.x;
   cuad->basico.infizq.y = cdro->basico.infizq.y;
   cuad->basico.supizq.x = cdro->basico.supizq.x;
   cuad->basico.supizq.y = cdro->basico.supizq.y;
   cuad->basico.supder.x = cdro->basico.supder.x;
   cuad->basico.supder.y = cdro->basico.supder.y;
   cuad->basico.infder.x = cdro->basico.infder.x;
   cuad->basico.infder.y = cdro->basico.infder.y;
   for (i = 0; i < NUMPAR; i++)
       cuad->basico.indob[i] = cdro->basico.indob[i];
   cuad->basico.figura = cdro->basico.figura;
   cuad->basico.pos = cdro->basico.pos;
   cuad->basico.ident = cdro->basico.ident;
   cuad->vecino = cdro->vecino;


* funcion que efectua la lista ligada de figuras vecinas */
truct lstcua *lstrec (ls,cdr)
truct lstcua *ls;
truct cuadrilatero *cdr;

   struct lstcua *p, *q;
```

```
    char *malloc();

    q = (struct lstcua *) malloc(sizeof (struct lstcua));
    obveci (&q->conten,cdr);
    q->prox = NULL;
    if (ls == NULL)
        ls = q;
    else {
        p = ls;
        while (p->prox != NULL)
            p = p->prox;
        p->prox = q;
    }
    return (ls);
}

/* funcion que obtiene el campo de informacion de la lista ligada */
/* de figuras vecinas */
obveci (cuad,cdro)
struct cuadrilatero *cuad, *cdro;
{
    int i;

    cuad->infizq.x = cdro->infizq.x;
    cuad->infizq.y = cdro->infizq.y;
    cuad->supizq.x = cdro->supizq.x;
    cuad->supizq.y = cdro->supizq.y;
    cuad->supder.x = cdro->supder.x;
    cuad->supder.y = cdro->supder.y;
    cuad->infder.x = cdro->infder.x;
    cuad->infder.y = cdro->infder.y;
    for (i = 0; i < NUMPAR; i++)
        cuad->indob[i] = cdro->indob[i];
    cuad->figura = cdro->figura;
    cuad->pos = cdro->pos;
    cuad->ident = cdro->ident;
}

/* funcion que encuentra una figura en la lista genereal a partir */
/* de su numero de referencia */
struct lstcdr *ubica (lsf,idfg)
struct lstcdr *lsf;
int idfg;
{
    struct lstcdr *p;

    if (lsf == NULL)
        return (NULL);
    else {
        p = lsf;
        while (p != NULL)
            if (p->nodo.basico.ident == idfg)
                return (p);
            else
                p = p->next;
```

```
    }
    printf ("Peligro. La figura no se encuentra en la lista\n");
    for (;;) ;
}

/* funcion que descompone una estructura de cuadrilatero en sus partes */
descol (cuad,p1,p2,p3,p4,fi,po)
struct cuadrilatero *cuad;
struct punto *p1,*p2,*p3,*p4;
int *fi,*po;
{
    *fi = cuad->figura ;
    *po = cuad->pos;
    p1->x = cuad->infizq.x ; p1->y = cuad->infizq.y;
    p2->x = cuad->supizq.x ; p2->y = cuad->supizq.y;
    p4->x = cuad->infder.x ; p4->y = cuad->infder.y;
    switch (cuad->figura) {
    case 0 : case 1 : case 2 : case 3 : case 4 : case 5 : {
        p3->x = cuad->supder.x ; p3->y = cuad->supder.y;
        } break;
    case 6 : case 7 :
        break;
    }


/* funcion que obtiene el campo tipo de una recta */
obtipo (pt1,pt2,t)
struct punto *pt1, *pt2;
int *t;
{
    if (pt1->y == pt2->y)
        *t = 0;
    else
        if (pt1->x == pt2->x)
            *t = 1;
    else
        if ((pt2->x - pt1->x) * (pt2->y - pt1->y) > 0)
            *t = 2;
    else
        *t = 3;
}

/* funcion que selecciona los extremos de los lados de un cuadrilatero */
selcuat (cuad,pt1,pt2,il)
struct cuadrilatero *cuad;
struct punto *pt1,*pt2;
int il;
{
    int fg,po;

    fg = cuad->figura ; po = cuad->pos;
    switch (il) {
    case 0 : {
        pt1->x = cuad->infizq.x ; pt1->y = cuad->infizq.y;
        pt2->x = cuad->supizq.x ; pt2->y = cuad->supizq.y;
```

```
        } break;
    case 1 : {
        pt1->x = cuad->supizq.x ; pt1->y = cuad->supizq.y;
        pt2->x = cuad->supder.x ; pt2->y = cuad->supder.y;
        if (((fg == 2) || (fg == 3) || (fg == 4)) && (po == 1)) {
            pt1->x = cuad->supder.x ; pt1->y = cuad->supder.y;
            pt2->x = cuad->supizq.x ; pt2->y = cuad->supizq.y;
            }
        } break;
    case 2 : {
        pt1->x = cuad->infder.x ; pt1->y = cuad->infder.y;
        pt2->x = cuad->supder.x ; pt2->y = cuad->supder.y;
        } break;
    case 3 : {
        pt1->x = cuad->infizq.x ; pt1->y = cuad->infizq.y;
        pt2->x = cuad->infder.x ; pt2->y = cuad->infder.y;
        if ((((fg == 2) || (fg == 3)) && (po == 3)) ||
            ((fg == 4) && (po == 1))) {
            pt1->x = cuad->infder.x ; pt1->y = cuad->infder.y;
            pt2->x = cuad->infizq.x ; pt2->y = cuad->infizq.y;
        }
        } break;
    }


  funcion que selecciona los extremos de los lados de un triangulo */
eltres (cuad,pt1,pt2,il)
truct cuadrilatero *cuad;
truct punto *pt1,*pt2;
nt il;

    int fg,po;

    fg = cuad->figura ; po = cuad->pos;
    switch (il) {
    case 0 : {
        pt1->x = cuad->infizq.x ; pt1->y = cuad->infizq.y;
        pt2->x = cuad->supizq.x ; pt2->y = cuad->supizq.y;
        } break;
    case 1 : {
        pt1->x = cuad->infder.x ; pt1->y = cuad->infder.y;
        pt2->x = cuad->supizq.x ; pt2->y = cuad->supizq.y;
        if ((po == 2) || ((fg == 6) && (po == 1))) {
            pt1->x = cuad->supizq.x ; pt1->y = cuad->supizq.y;
            pt2->x = cuad->infder.x ; pt2->y = cuad->infder.y;
        }
        } break;
    case 2 : {
        pt1->x = cuad->infizq.x ; pt1->y = cuad->infizq.y;
        pt2->x = cuad->infder.x ; pt2->y = cuad->infder.y;
        if ((fg == 7) && (po == 3)) {
            pt1->x = cuad->infder.x ; pt1->y = cuad->infder.y;
            pt2->x = cuad->infizq.x ; pt2->y = cuad->infizq.y;
        }
        } break;
```

```
    }


* funcion que compara si dos lados de dos figuras tienen algo en comun. */
* para anadir un nuevo vecino a la lista */
ompara (ps1,ps2,pv1,pv2,tv)
truct punto *ps1,*ps2,*pv1,*pv2;
nt tv;

    int da,db,res;

    switch (tv) {
    case 0 :
        if ((ps1->y == pv1->y) && (((ps1->x >= pv1->x) && (ps2->x <= pv1->x))
            || ((ps1->x < pv1->x) && (ps2->x > pv2->x)) ||
            ((ps1->x < pv1->x) && (ps2->x <= pv2->x) && (ps2->x > pv1->x)) ||
            ((ps1->x >= pv1->x) && (ps1->x < pv2->x) && (ps2->x > pv2->x))))
            res = 1;
        else
            res = 0;
        break;
    case 1 :
        if ((ps1->x == pv1->x) && (((ps1->y >= pv1->y) && (ps2->y <= pv2->y))
            || ((ps1->y < pv1->y) && (ps2->y > pv2->y)) ||
            ((ps1->y < pv1->y) && (ps2->y <= pv2->y) && (ps2->y > pv1->y)) ||
            ((ps1->y >= pv1->y) && (ps1->y < pv2->y) && (ps2->y > pv2->y))))
            res = 1;
        else
            res = 0;
       . break;
    case 2 : {
        da = ps1->y - ps1->x + pv1->x - pv1->y;
        db = ps2->y - ps2->x + pv1->x - pv1->y;
        if ((da == 0) && (db == 0) &&
            (((ps1->x >= pv1->x) && (ps2->x <= pv2->x)) ||
            ((ps1->x < pv1->x) && (ps2->x > pv2->x)) ||
            ((ps1->x < pv1->x) && (ps2->x <= pv2->x) && (ps2->x > pv1->x)) ||
            ((ps1->x >= pv1->x) && (ps1->x < pv2->x) && (ps2->x > pv2->x))))
            res = 1;
        else
            res = 0;
        }
        break;
    case 3 : {
        da = ps1->y + ps1->x - pv1->x - pv1->y;
        db = ps2->y + ps2->x - pv1->x - pv1->y;
        if ((da == 0) && (db == 0) &&
            (((ps1->y >= pv1->y) && (ps2->y <= pv2->y)) ||
            ((ps1->y < pv1->y) && (ps2->y > pv2->y)) ||
            ((ps1->y < pv1->y) && (ps2->y <= pv2->y) && (ps2->y > pv1->y)) ||
            ((ps1->y >= pv1->y) && (ps1->y < pv2->y) && (ps2->y > pv2->y))))
            res = 1;
        else
            res = 0;
        }
```

```
        break;
    }
    return (res);
}

/* funcion que recorre la lista general de figuras imprimiendo su contenido
muestra (ar,ls)
FILE *ar;
struct lstcdr *ls;
{
    struct lstcdr *p;

    p = ls;
    while (p != NULL) {
        imprime (ar,&p->nodo.basico,1);
        ensena (ar,p->nodo.vecino);
        p = p->next;
    }
}

/* funcion que recorre la lista de figuras vecinas imprimiendo su contenido
ensena (f,ls)
FILE *f;
struct lstcua *ls;
{
    struct lstcua *p;

    fprintf (f,"\nLISTA DE FIGURAS VECINAS\n");
    fprintf (f,"\nFIGURAS N ");
    p = ls;
    while (p != NULL) {
        fprintf (f,"%5d",p->conten.ident);
        p = p->prox;
    }
    fprintf (f,"\n");
}

/* funcion que imprime los datos de un cuadrilatero de la grafica */
imprime (f,rec,ind)
FILE *f;
struct cuadrilatero *rec;
int ind;
{
    int fg,i;

    fg = rec->figura;
    fprintf (f,"\nFIGURA DE LA GRAFICA N ") ; fprintf (f,"%d\n",rec->ident);
    fprintf (f,"\nPUNTOS\n");
    if ((fg == 6) || (fg == 7))
        fprintf (f,"Punto 1 = ");
    else
        fprintf (f,"Punto 1 (Inferior izquierdo) = ");
    fprintf (f,"%5d",rec->infizq.x);
    fprintf (f,"%5d\n",rec->infizq.y);
    if ((fg == 6) || (fg == 7))
```

```
    fprintf (f,"Punto 2 = ");
else
    fprintf (f,"Punto 2 (Superior izquierdo) = ");
fprintf (f,"%5d",rec->supizq.x);
fprintf (f,"%5d\n",rec->supizq.y);
if ((fg != 6) && (fg != 7)) {
    fprintf (f,"Punto 3 (Superior derecho) = ");
    fprintf (f,"%5d",rec->supder.x);
    fprintf (f,"%5d\n",rec->supder.y);
}
if ((fg == 6) || (fg == 7))
    fprintf (f,"Punto 3 = ");
else
    fprintf (f,"Punto 4 (Inferior derecho) = ");
fprintf (f,"%5d",rec->infder.x);
fprintf (f,"%5d\n",rec->infder.y);
fprintf (f,"Figura tipo = ");
fprintf (f,"%5d\n",rec->figura);
fprintf (f,"Posicion = ");
fprintf (f,"%5d\n",rec->pos);
if (ind == 1) {
    fprintf (f,"\nLADOS OBSTACULOS DE LA FIGURA\n\n");
    switch (fg) {
    case 0 : case 1 : case 2 : case 3 : case 4 : case 5 :
        for (i = 0; i < NUMVER; i++)
            if (rec->indob [i] == 1)
                switch (i) {
                case 0 :
                    fprintf (f,"Obstaculo en lado p1-p2\n");
                    break;
                case 1 :
                    fprintf (f,"Obstaculo en lado p2-p3\n");
                    break;
                case 2 :
                    fprintf (f,"Obstaculo en lado p4-p3\n");
                    break;
                case 3 :
                    fprintf (f,"Obstaculo en lado p1-p4\n");
                    break;
                }
        break;
    case 6 : case 7 :
        for (i = 0; i < NUMVER - 1; i++)
            if (rec->indob [i] == 1)
                switch (i) {
                case 0 :
                    fprintf (f,"Obstaculo en lado p1-p2\n");
                    break;
                case 1 :
                    fprintf (f,"Obstaculo en lado p3-p2\n");
                    break;
                case 2 :
                    fprintf (f,"Obstaculo en lado p1-p3\n");
                    break;
                }
```

```
            break;
        }
    }
}

/* funcion que determina si un numero es par impar */
/* si es par retorna 1, si es impar retorna 0 */
espar (x)
int x;
{
    int res = 0;

    if (x/2*2 == x)
        res = 1;
    return (res);
}
```

2. PROGRAMA PROPAGA

```
#include "include\time.h"
#include "include\stdio.h"
#include "estruc.h"
#include "externo.h"

int ndn;

/* programa que calcula la ruta de longitud minima entre dos puntos */
/* de la grafica */
main ()
{
    FILE *fp, *fg, *fr, *fopen();
    struct lstcdr *base = NULL, *api, *apf, *pinicial();
    struct queue *frente = NULL, *clearqu();
    struct cerrado *close = NULL, *lstran(), *clearcl();
    struct ranura inicial,antec,final,penul;
    struct punto pi,pf;
    int si,c,c1,c2,nor,dx,dy,dist,dist1,np,nr,fclose();
    long t;

    fr = fopen ("trayec.dat","w");
    printf ("\nDETERMINACION DE LA RUTA DE LONGITUD MINIMA ENTRE\n");
    printf ("DOS PUNTOS DEL PLANO EN PRESENCIA DE OBSTACULOS\n");
    printf ("\nALGORITMO DE PROPAGACION DE RANURAS (R.Barrera-D.Andrade)\n")
    fprintf (fr,"\nDETERMINACION DE LA RUTA DE LONGITUD MINIMA ENTRE\n");
    fprintf (fr,"DOS PUNTOS DEL PLANO EN PRESENCIA DE OBSTACULOS\n");
    fprintf (fr,"\nALGORITMO DE PROPAGACION DE RANURAS (R.Barrera-D.Andrade)
    fp = fopen ("grafica.dat","r");
    base = leegra (base,fp);
    fclose (fp) ;
    printf ("\nDATOS DE LA GRAFICA CARGADOS\n");
    printf ("\nDesea impresion de los datos de la grafica ? (s/n) ");
    c = getchar();
    if ((c == 's') || (c == 'S')) {
        fg = fopen ("figura.dat","w");
        muestra (fg,base);
        fclose (fg);
    }
    do {
        (struct cerrado *) close = NULL;
        (struct queue *) frente = NULL;
        np = nr = 0;
        printf ("\nDesea impresion de todas las rutas");
        printf (" desde 's' hasta 't' ? (s/n) ");
        getchar() ; c2 = getchar();
        printf ("\nDesea despliege de la generacion de ranuras ? (s/n) ");
        getchar() ; c1 = getchar();.
        printf ("\nELIGE LA OPCION DE CALCULO PARA LA DISTANCIA\n");
        printf ("\n0. Con norma L1.\n");
        printf ("1. Con norma L infinito.\n");
        printf ("\nOpcion : ") ; scanf ("%d",&nor);
        while ((nor != 0) && (nor != 1)) {
            printf ("\nOpcion no valida, por favor ingrese nuevamente");
```

```
    printf ("\nOpcion : ") ; scanf ("%d",&nor); }
printf ("\nCOORDENADAS DE LOS PUNTOS INICIAL Y FINAL\n");
printf ("\nPunto inicial\n");
printf ("Abscisa = ") ; scanf ("%d",&pi.x);
printf ("Ordenada = ") ; scanf ("%d",&pi.y);
printf ("\nPunto final\n");
printf ("Abscisa = ") ; scanf ("%d",&pf.x);
printf ("Ordenada = ") ; scanf ("%d",&pf.y);
fprintf (fr,"\nCOORDENADAS DE LOS PUNTOS INICIAL Y FINAL\n");
fprintf (fr,"\nPunto inicial\n");
fprintf (fr,"Abscisa = ") ; fprintf (fr,"%d\n",pi.x);
fprintf (fr,"Ordenada = ") ; fprintf (fr,"%d\n",pi.y);
fprintf (fr,"\nPunto final\n");
fprintf (fr,"Abscisa = ") ; fprintf (fr,"%d\n",pf.x);
fprintf (fr,"Ordenada = ") ; fprintf (fr,"%d\n",pf.y);
time (&t);
printf ("\nLa hora actual es %s\n",ctime (&t));
fprintf (fr,"\nLa hora actual es %s\n",ctime (&t));
api = pinicial (base,&pi);
inicial.refer.x = pi.x ; inicial.refer.y = pi.y;
inicial.dista = 0 ; inicial.idfig = api->nodo.basico.ident;
inicial.desde = inicial.rtipo = -1 ; inicial.previo = NULL;
copranu (&antec,&inicial);
apf = pinicial (base,&pf);
if (apf->nodo.basico.ident == api->nodo.basico.ident) {
    printf ("\nNo existen obstaculos entre los puntos inicial y final.\n
    printf ("La distancia entre los puntos inicial y final ");
    fprintf (fr,"\nLa distancia entre los puntos inicial y final ");
    dx = abs (pf.x - pi.x) ; dy = abs (pf.y - pi.y);
    switch (nor) {
    case 0 : {
        dist = dx + dy;
        printf ("medida con norma L1 es : %d\n",dist);
        fprintf (fr,"medida con norma L1 es : %d\n",dist);
        } break;
    case 1 : {
        dist = max (dx,dy);
        printf ("medida con norma Loo es : %d\n",dist);
        fprintf (fr,"medida con norma Loo es : %d\n",dist);
        } break;
    }
}
else {
    ranfin (apf,&pf,&final);
    if ((c1 == 's') || (c1 == 'S'))
        printf ("\n                PROPAGACION DE RANURAS\n");
    close = lstran (close,&antec) ; nr++;
    frente = disvec (frente,close,api,&antec,nor,c1);
    while ((antec.idfig != final.desde) && (frente != NULL)) {
        frente = recupera (frente,&antec);
        if ((misma (close,&antec)) == 0)
            nr++;
        close = lstran (close,&antec);
        api = ubica (base,antec.idfig);
        frente = disvec (frente,close,api,&antec,nor,c1);
```

```
        }
        if (frente == NULL) {
            printf ("\nNo existe ruta entre los puntos :\n");
            printf ("\nInicial :%5d%5d\n",pi.x,pi.y);
            printf ("Final :%5d%5d\n",pf.x,pf.y);
            fprintf (fr,"\nNo existe ruta entre los puntos :\n");
            fprintf (fr,"\nInicial :%5d%5d\n",pi.x,pi.y);
            fprintf (fr,"Final :%5d%5d\n",pf.x,pf.y);
        }
        else {
            time (&t);
            printf ("\nLa hora actual es %s\n",ctime (&t));
            fprintf (fr,"\nLa hora actual es %s\n",ctime (&t));
            printf ("\nNumero de ranuras propagadas = %d\n",nr);
            fprintf (fr,"\nNumero de ranuras propagadas = %d\n",nr);
            caldis (&antec,&final,&dist,nor);
            fprintf (fr,"\nAcceso al punto final con distancia : %d\n",dist)
            copranu (&penul,&antec);
            if ((c2 == 's') || (c2 == 'S'))
                impruta (&inicial,&final,&penul,np,dist,nor,fr);
            while ((antec.dista <= dist) && (frente != NULL)) {
                frente = recupera (frente,&antec);
                if ((misma (close,&antec)) == 0)
                    nr++;
                close = lstran (close,&antec);
                if (antec.idfig == final.desde) {
                    caldis (&antec,&final,&dist1,nor);
                    fprintf (fr,"\nAcceso al punto final con distancia :");
                    fprintf (fr," %d\n",dist1);
                    if ((c2 == 's') || (c2 == 'S'))
                        impruta (&inicial,&final,&penul,np,dist1,nor,fr);
                    if (dist1 < dist) {
                        dist = dist1;
                        copranu (&penul,&antec);
                    }
                }
                api = ubica (base,antec.idfig);
                frente = disvec (frente,close,api,&antec,nor,c1);
            }
            time (&t);
            printf ("\nLa hora actual es %s\n",ctime (&t));
            fprintf (fr,"\nLa hora actual es %s\n",ctime (&t));
            printf ("\n*** FIN DEL PROCESO DE ENRUTAMIENTO ***\n");
            printf ("\nNumero de ranuras propagadas = %d\n",nr);
            fprintf (fr,"\n*** FIN DEL PROCESO DE ENRUTAMIENTO ***\n");
            fprintf (fr,"\nNumero de ranuras propagadas = %d\n",nr);
            close = lstran (close,&final);
            impruta (&inicial,&final,&penul,np,dist,nor,fr);
            frente = clearqu (frente) ; close = clearcl (close);
        }
    }
    printf ("\nDesea determinar la ruta entre otros puntos ? (s/n) ");
    getchar() ; si = getchar();
    } while ((si == 's') || (si == 'S'));
fclose (fr); }
```

```
/* funcion que imprime la ruta */
pruta (inic,fin,pen,np,dist,nor,ar)
truct ranura *inic,*fin,*pen;
nt np,dist,nor;
ILE *ar;
{
   struct ranura wedge,*apr;
   struct pila *ruta = NULL, *push(), *pop();

   fin->previo = pen ; fin->dista = dist;
   ruta = push (ruta,fin);
   apr = fin->previo;
   do {
       ruta = push (ruta,apr);
       apr = apr->previo;
       } while (apr->desde != -1);
    ruta = push (ruta,inic);
    printf ("\nRUTA ENTRE EL PUNTO INICIAL Y EL PUNTO FINAL\n");
    printf ("\n      PUNTO      ABSCISA      ORDENADA\n\n") ;
    fprintf (ar,"\nRUTA ENTRE EL PUNTO INICIAL Y EL PUNTO FINAL\n");
    fprintf (ar,"\n      PUNTO      ABSCISA      ORDENADA\n\n") ;
    do {
        ruta = pop (ruta,&wedge);
        printf ("%10d%11d%12d\n",++np,wedge.refer.x,wedge.refer.y);
        fprintf (ar,"%10d%11d%12d\n",np,wedge.refer.x,wedge.refer.y);
        } while (ruta != NULL);
     printf ("\nLa distancia entre los puntos inicial y final ");
     fprintf (ar,"\nLa distancia entre los puntos inicial y final ");
     switch (nor) {
     case 0 :
         printf ("medida con norma L1 es : %d\n",fin->dista);
         fprintf (ar,"medida con norma L1 es : %d\n",fin->dista);
         break;
     case 1 :
         printf ("medida con norma Loo es : %d\n",fin->dista);
         fprintf (ar,"medida con norma Loo es : %d\n",fin->dista);
         break;
     }
}

/* funcion que determina cual es la region geometrica donde se */
/* encuentra el punto inicial para la busqueda */
truct lstcdr *pinicial (ls,pt)
truct lstcdr *ls;
truct punto *pt;

   struct lstcdr *p;
   struct punto p1,p2,p3,p4;
   int fig,po,ahi;

   if (ls == NULL)
       return (NULL);
   else {
       p = ls;
       while (p != NULL) {
```

```
            descol (&p->nodo.basico,&p1,&p2,&p3,&p4,&fig,&po);
            switch (fig) {
            case 0 :
                ahi = figura0 (&p1,&p2,&p3,&p4,pt);
                break;
            case 1 : case 6 :
                ahi = figur16 (&p1,&p2,&p3,&p4,pt,po);
                break;
            case 2 :
                ahi = figura2 (&p1,&p2,&p3,&p4,pt,po);
                break;
            case 3 :
            case 7 :
                ahi = figur37 (&p1,&p2,&p3,&p4,pt,po);
                break;
            case 4 :
                ahi = figura4 (&p1,&p2,&p3,&p4,pt,po);
                break;
            case 5 :
                ahi = figura5 (&p1,&p2,&p3,&p4,pt,po);
                break;
            }
            if (ahi == 1)
                return (p);
            else
                p = p->next;
        }
        printf("peligro. el punto no esta en las figuras.\n");
        for (;;) ;
    }

/* funcion que considera que el punto inicial cae en un rectangulo */
figura0 (p1,p2,p3,p4,pi)
struct punto *p1,*p2,*p3,*p4,*pi;
{
    int res = 0;

    if ((pi->x >= p1->x) && (pi->x <= p3->x) &&
        (pi->y >= p1->y) && (pi->y <= p3->y))
        res = 1;
    return (res);
}

/* funcion que considera que el punto inicial cae en un trapecio o */
/* triangulo rectangulo izquierdo */
figur16 (p1,p2,p3,p4,pa,po)
struct punto *p1,*p2,*p3,*p4,*pa;
int po;
{
    int di,res = 0;

    switch (po) {
    case 0 : {
        di = pa->y + pa->x - p4->y - p4->x;
```

```
        if ((pa->x >= p1->x) && (di < 0) &&
            (pa->y >= p1->y) && (pa->y <= p2->y))
            res = 1;
        } break;
    case 1 : {
        di = pa->y - pa->x + p1->x - p1->y;
        if ((pa->x >= p1->x) && (pa->x <= p4->x) && (di >= 0) &&
            (pa->y <= p2->y))
            res = 1;
        } break;
    case 2 : {
        di = pa->y + pa->x - p1->y - p1->x;
        if ((di >= 0) && (pa->x <= p4->x) &&
            (pa->y >= p1->y) && (pa->y <= p2->y))
            res = 1;
        } break;
    case 3 : {
        di = pa->y - pa->x + p2->x - p2->y;
        if ((pa->x >= p1->x) && (di < 0) &&
            (pa->x <= p4->x) && (pa->y >= p1->y))
            res = 1;
        } break;
    }
    return (res);


/* funcion que considera que el punto inicial cae en un trapecio */
/* rectangulo derecho */
figura2 (p1,p2,p3,p4,pa,po)
struct punto *p1,*p2,*p3,*p4,*pa;
int po;
{
    int di,res = 0;

    switch (po) {
    case 0 : {
        di = pa->y - pa->x + p1->x - p1->y;
        if ((di <= 0) && (pa->x <= p3->x) &&
            (pa->y >= p1->y) && (pa->y <= p3->y))
            res = 1;
        } break;
    case 1 : {
        di = pa->y + pa->x - p3->x - p3->y;
        if ((pa->x >= p1->x) && (pa->x <= p3->x) &&
            (di < 0) && (pa->y >= p1->y))
            res = 1;
        } break;
    case 2 : {
        di = pa->y - pa->x + p4->x - p4->y;
        if ((pa->x >= p1->x) && (di > 0) &&
            (pa->y >= p1->y) && (pa->y <= p3->y))
            res = 1;
        } break;
    case 3 : {
        di = pa->y + pa->x - p4->x - p4->y;
```

```
        if ((pa->x >= p1->x) && (di >= 0) &&
            (pa->x <= p4->x) && (pa->y <= p3->y))
            res = 1;
        } break;
    }
    return (res);
}

/* funcion que considera que el punto inicial cae en un trapecio o */
/* triangulo rectangulo isosceles */
figur37 (p1,p2,p3,p4,pa,po)
struct punto *p1,*p2,*p3,*p4,*pa;
int po;
{
    int di,di1,res = 0;

    switch (po) {
    case 0 : {
        di = pa->y + pa->x - p4->y - p4->x;
        di1 = pa->y - pa->x + p1->x - p1->y;
        if ((di < 0) && (di1 <= 0) &&
            (pa->y >= p1->y) && (pa->y <= p2->y))
            res = 1;
        } break;
    case 1 : {
        di = pa->y - pa->x + p1->x - p1->y;
        di1 = pa->y + pa->x - p3->x - p3->y;
        if ((pa->x >= p1->x) && (pa->x <= p4->x) &&
            (di >= 0) && (di1 < 0))
            res = 1;
        } break;
    case 2 : {
        di = pa->y + pa->x - p1->y - p1->x;
        di1 = pa->y - pa->x + p4->x - p4->y;
        if ((di >= 0) && (di1 > 0) &&
            (pa->y >= p1->y) && (pa->y <= p2->y))
            res = 1;
        } break;
    case 3 : {
        di = pa->y - pa->x + p2->x - p2->y;
        di1 = pa->y + pa->x - p4->x - p4->y;
        if ((pa->x >= p1->x) && (pa->x <= p4->x) &&
            (di < 0) && (di1 >= 0))
            res = 1;
        } break;
    }
    return (res);
}

/* funcion que considera que el punto inicial cae en un romboide agudo */
figura4 (p1,p2,p3,p4,pa,po)
struct punto *p1,*p2,*p3,*p4,*pa;
int po;
{
    int di,di1,res = 0;
```

```
    switch (po) {
    case 0 : {
        di = pa->y - pa->x - p4->y + p4->x;
        di1 = pa->y - pa->x + p1->x - p1->y;
        if ((di1 <= 0) && (di > 0) &&
            (pa->y >= p1->y) && (pa->y <= p2->y))
            res = 1;
        } break;
    case 1 : {
        di = pa->y + pa->x - p3->y - p3->x;
        di1 = pa->y + pa->x - p4->x - p4->y;
        if ((pa->x >= p1->x) && (di1 >= 0) &&
            (pa->x <= p4->x) && (di < 0))
            res = 1;
        } break;
    }
    return (res);
}

/* funcion que considera que el punto inicial cae en un romboide obtuso */
figura5 (p1,p2,p3,p4,pa,po)
struct punto *p1,*p2,*p3,*p4,*pa;
int po;

    int di,di1,res = 0;

    switch (po) {
    case 0 : {
        di = pa->y + pa->x - p4->y - p4->x;
        di1 = pa->y + pa->x - p1->x - p1->y;
        if ((di1 >= 0) && (di < 0) &&
            (pa->y >= p1->y) && (pa->y <= p2->y))
            res = 1;
        } break;
    case 1 : {
        di = pa->y - pa->x - p1->y + p1->x;
        di1 = pa->y - pa->x + p2->x - p2->y;
        if ((pa->x >= p1->x) && (pa->x <= p4->x) &&
            (di1 < 0) && (di >= 0))
            res = 1;
        } break;
    }
    return (res);
}

/* funcion que calcula la distancia hasta la ranura final */
caldis (ans,fin,d,norma)
struct ranura *ans,*fin;
int *d,norma;
{
    switch (norma) {
    case 0 :
        *d = ans->dista + abs (fin->refer.y - ans->refer.y) +
             abs (fin->refer.x - ans->refer.x);
```

```
        break;
    case 1 :
        *d = ans->dista + max (abs (fin->refer.x - ans->refer.x)
             abs (fin->refer.y - ans->refer.y));
        break;
    }
}

/* funcion que genera la ranura final */
anfin (p,pt,wdg)
truct lstcdr *p;
truct punto *pt;
truct ranura *wdg;
{
    struct punto p1,p2;
    int i,tr,esta,si = 0;

    switch (p->nodo.basico.figura) {
    case 0 : case 1 : case 2 : case 3 : case 4 : case 5 : {
        for (i = 0; ((i < NUMPAR) && (si == 0)); i++) {
            selcuat (p,&p1,&p2,i);
            switch (i) {
            case 0 : case 2 :
                esta = enlizde (&p1,&p2,pt,&tr);
                break;
            case 1 : case 3 :
                esta = enlinsu (&p1,&p2,pt,&tr);
                break;
            }
            if (esta == 1) {
                formran (p,wdg,pt,tr);
                si = 1;
            }
        }
        if (si == 0) {
            switch (p->nodo.basico.pos) {
            case 0 : case 2 :
                tr = 0;
                break;
            case 1 : case 3 :
                tr = 1;
                break;
            }
            formran (p,wdg,pt,tr);
        }
        } break;
    case 6 : case 7 : {
        for (i = 0; ((i < NUMVER - 1) && (si == 0)); i++) {
            seltres (p,&p1,&p2,i);
            switch (i) {
            case 0 :
                esta = enlizde (&p1,&p2,pt,&tr);
                break;
            case 1 :
                esta = enlado2 (&p1,&p2,pt,&tr);
```

```
                break;
            case 2 :
                esta = enlado3 (&p1,&p2,pt,&tr);
                break;
            }
            if (esta == 1) {
                formran (p,wdg,pt,tr);
                si = 1;
            }
        }
        if (si == 0) {
            switch (p->nodo.basico.pos) {
            case 0 : case 2 :
                tr = 0;
                break;
            case 1 : case 3 :
                tr = 1;
                break;
            }
            formran (p,wdg,pt,tr);
        }
        } break;
    }
}

/* funcion que construye una ranura en base a ciertos datos */
formran (ap,ran,pf,ti)
struct lstcdr *ap;
struct ranura *ran;
struct punto *pf;
int ti;

   ran->refer.x = pf->x ; ran->refer.y = pf->y;
   ran->rtipo = ti ; ran->dista = 0 ;
   ran->desde = ap->nodo.basico.ident ; ran->idfig = -1;
   ran->previo = NULL;
}

/* funcion auxiliar a enlado que analiza si el punto se encuentra */
/* en los lados izquierdo o derecho */
enlizde (p1,p2,pf,t)
struct punto *p1,*p2,*pf;
int *t;

   int tl,res = 0;

   obtipo (p1,p2,&tl);
   switch (tl) {
   case 1 :
       if ((pf->x == p1->x) && (pf->y >= p1->y) && (pf->y <= p2->y))
           res = 1;
       break;
   case 2 :
       if ((pf->y >= p1->y) && (pf->y <= p2->y) &&
           (pf->y - pf->x + p1->x - p1->y == 0))
```

```
            res = 1;
        break;
    case 3 :
        if ((pf->y >= p1->y) && (pf->y <= p2->y) &&
            (pf->y + pf->x - p1->x - p1->y == 0))
            res = 1;
        break;
    }
    *t = tl;
    return (res);
}

/* funcion auxiliar a enlado que analiza si el punto se encuentra */
/* en los lados inferior o superior */
enlinsu (p1,p2,pf,t)
struct punto *p1,*p2,*pf;
int *t;
{
    int tl,res = 0;

    obtipo (p1,p2,&tl);
    switch (tl) {
    case 0 :
        if ((pf->y == p1->y) && (pf->x >= p1->x) && (pf->x <= p2->x))
            res = 1;
        break;
    case 2 :
        if ((pf->x >= p1->x) && (pf->x <= p2->x) &&
            (pf->y - pf->x + p1->x - p1->y == 0))
            res = 1;
        break;
    case 3 :
        if ((pf->x >= p1->x) && (pf->x <= p2->x) &&
            (pf->y + pf->x - p1->x - p1->y == 0))
            res = 1;
        break;
    }
    *t = tl;
    return (res);


* funcion auxiliar a enlado que analiza si el punto se encuentra */
* en el segundo lado del triangulo */
nlado2 (p1,p2,pf,t)
truct punto *p1,*p2,*pf;
nt *t;

   int tl,res = 0;

   obtipo (p1,p2,&tl);
   switch (tl) {
   case 0 :
       if ((pf->y == p1->y) && (pf->x >= p1->x) && (pf->x <= p2->x))
           res = 1;
       break;
```

```
    case 1 :
        if ((pf->x == p1->x) && (pf->y >= p1->y) && (pf->y <= p2->y))
            res = 1;
        break;
    case 3 :
        if ((pf->y >= p1->y) && (pf->y <= p2->y) &&
            (pf->y + pf->x - p1->x - p1->y == 0))
            res = 1;
        break;
    }
    *t = tl;
    return (res);


* funcion auxiliar a enlado que analiza si el punto se encuentra */
* en el tercer lado del triangulo */
nlado3 (p1,p2,pf,t)
truct punto *p1,*p2,*pf;
nt *t;

    int tl,res = 0;

    obtipo (p1,p2,&tl);
    switch (tl) {
    case 0 :
        if ((pf->y == p1->y) && (pf->x >= p1->x) && (pf->x <= p2->x))
            res = 1;
        break;
    case 1 :
        if ((pf->x == p1->x) && (pf->y >= p1->y) && (pf->y <= p2->y))
            res = 1;
        break;
    case 2 :
        if ((pf->x >= p1->x) && (pf->x <= p2->x) &&
            (pf->y - pf->x + p1->x - p1->y == 0))
            res = 1;
        break;
    case 3 :
        if ((pf->y >= p1->y) && (pf->y <= p2->y) &&
            (pf->y + pf->x - p1->x - p1->y == 0))
            res = 1;
        break;
    }.
    *t = tl;
    return (res);


* funcion que forma la lista ligada de ranuras propagadas */
truct cerrado *lstran (lr,r)
truct cerrado *lr;
truct ranura *r;

    struct cerrado *p,*q;
    char *malloc();
```

```
    q = (struct cerrado *) malloc(sizeof(struct cerrado));
    copranu (&q->visit,r);
    q->proxi = NULL;
    if (lr == NULL)
        lr = q;
    else {
        p = lr;
        while (p->proxi != NULL)
            p = p->proxi;
        p->proxi = q;
    }
    return (lr);


* funcion que recupera un elemento del frente de la cola de ranuras */
truct queue *recupera (lq,ran)
truct queue *lq;
truct ranura *ran;

    struct queue *p;

    if (lq == NULL) {
        printf ("No existen mas ranuras para propagarse\n");
        return (NULL);
    }
    else {
        p = lq;
        copranu (ran,&p->info);
        lq = lq->sigui;
        free ((char *) p);
        return (lq);
    }


* funcion que empuja un elemento en la pila de ruta */
truct pila *push (t,wdg)
truct pila *t;
truct ranura *wdg;

    struct pila *p;

    p = (struct pila *) malloc(sizeof(struct pila));
    copranu (&p->infor,wdg);
    p->sigu = t;
    t = p;
    return (t);


* funcion que recupera un elemento del tope de la pila */
truct pila *pop (t,wdg)
truct pila *t;
truct ranura *wdg;

    struct pila *p;
```

```
    if (t == NULL)
        return (NULL);
    else {
        p = t;
        t = p->sigu;
        copranu (wdg,&p->infor);
        free ((char *) p);
        return (t);
    }
}

/* funcion que limpia la cola prioritaria */
struct queue *clearqu (lc)
struct queue *lc;
{
    struct queue *p;

    if (lc == NULL)
        return (lc);
    else {
        p = lc;
        do {
            free ((char *) p->info.previo);
            free ((char *) p);
            p = p->sigui;
            } while (p != NULL);
    }
    return (lc);
}

/* funcion que limpia la lista de ranuras propagadas */
struct cerrado *clearcl (lc)
struct cerrado *lc;
{
    struct cerrado *p;

    if (lc == NULL)
        return (lc);
    else {
        p = lc;
        do {
            free ((char *) p->visit.previo);
            free ((char *) p);
            p = p->proxi;
            } while (p != NULL);
    }
    return (lc);
}

/* funcion que determina si un punto pertenece a una misma ranura */
misma (lc,wdg)
struct cerrado *lc;
struct ranura *wdg;
{
    struct cerrado *p;
```

```
int res = 0;

if (lc == NULL)
    return (res);
else {
    p = lc;
    while (p != NULL)
        if ((p->visit.desde == wdg->desde) &&
            (p->visit.idfig == wdg->idfig) &&
            (p->visit.rtipo == wdg->rtipo)) {
            res = 1;
            return (res);
        }
        else
            p = p->proxi;
    return (res);
}
```

```
struct ranura desce [3];

/* funcion que calcula las distancias desde un punto de referencia */
/* de una figura hacia las figuras vecinas */
struct queue *disvec (lq,close,ap,corr,norma,c)
struct queue *lq;
struct cerrado *close;
struct lstcdr *ap;
struct ranura *corr;
int norma,c;
{
    struct queue *substit();
    struct lstcua *p;
    struct punto pt1,pt2;
    int i,j,t,si,idv;

    if (ap->nodo.vecino == NULL)
        printf ("No existe propagacion de ranuras\n");
    else {
        if ((c == 's') || (c == 'S')) {
            printf ("\nRANURA QUE SE PROPAGA\n");
            if (corr->desde == -1)
                printf ("\nPaso : Inicio ");
            else
                printf ("\nPaso : %4d ",corr->desde);
            printf ("%4d\n",corr->idfig);
            printf ("Punto de referencia :%5d%5d\n",corr->refer.x,corr->refer.y);
            printf ("\n                    PUNTO REFERENCIA\n");
            printf ("DESDE    HACIA    ABSCISA    ORDENADA    DISTANCIA\n\n");
        }
        p = ap->nodo.vecino;
        while (p != NULL) {
          si = 0 ; idv = p->conten.ident;
          switch (ap->nodo.basico.figura) {
          case 0 : case 1 : case 2 : case 3 : case 4 : case 5 :
              for (j = 0; ((j < NUMVER) && (si == 0)); j++)
                  if (ap->nodo.basico.indob[j] != 1) {
                      selcuat (&ap->nodo.basico,&pt1,&pt2,j);
                      obtipo (&pt1,&pt2,&t);
                      si = calcran (p,corr,&pt1,&pt2,t,norma,idv);
                      if (si == 1)
                          for (i = 0; i < 3; i++) {
                              if ((desce[i].dista != -1) &&
                                  ((encola (lq,&desce[i])) == 1))
                                      desce[i].dista = -1;
                              if (desce[i].dista != -1)
                                  lq = substit (lq,&desce[i]);
                              if (((comprue (&desce[i].close)) == 0) &&
                                  (desce[i].dista != -1)) {
                                  if ((c == 's') || (c == 'S'))
                                      escribe (&desce[i]);
                                  lq = inserta (lq,&desce[i]);
                              }
                          }
                  }
```

```
            break;
        case 6 : case 7 :
            for (j = 0; ((j < NUMVER - 1) && (si == 0)); j++)
                if (ap->nodo.basico.indob[j] != 1) {
                    seltres (&ap->nodo.basico,&pt1,&pt2,j);
                    obtipo (&pt1,&pt2,&t);
                    si = calcran (p.corr,&pt1,&pt2,t,norma.idv);
                    if (si == 1)
                        for (i = 0; i < 3; i++) {
                            if ((desce[i].dista != -1) &&
                                ((encola (lq,&desce[i])) == 1))
                                    desce[i].dista = -1;
                            if (desce[i].dista != -1)
                                lq = substit (lq,&desce[i]);
                            if (((comprue (&desce[i].close)) == 0) &&
                                (desce[i].dista != -1)) {
                                if ((c == 's') || (c == 'S'))
                                    escribe (&desce[i]);
                                lq = inserta (lq,&desce[i]);
                            }
                        }
                }
            break;
        }
        p = p->prox;
    }
}
return (lq);


* funcion auxiliar a disvec que calcula las ranuras correspondientes a */
* la figura que se esta propagando */
alcran (pv,rcor,pb1,pb2,tb,nor,idv)
truct lstcua *pv;
truct ranura *rcor;
truct punto *pb1,*pb2;
nt tb,nor,idv;

{
    struct punto pa1,pa2;
    int ta,i,si = 0;

    switch (pv->conten.figura) {
    case 0 : case 1 : case 2 : case 3 : case 4 : case 5 :
        for (i = 0; ((i < NUMVER) && (si == 0)); i++)
            if (pv->conten.indob[i] != 1) {
                selcuat (&pv->conten,&pa1,&pa2,i);
                obtipo (&pa1,&pa2,&ta);
                if ((ta == tb) &&
                    ((si = compara (pb1,pb2,&pa1,&pa2,ta)) == 1))
                    calcula (rcor,&pa1,&pa2,pb1,pb2,idv,ta,nor);
            }
        break;
    case 6 : case 7 :
        for (i = 0; ((i < NUMVER - 1) && (si == 0)); i++)
            if (pv->conten.indob[i] != 1) {
```

```
                seltres (&pv->conten,&pa1,&pa2,i);
                obtipo (&pa1,&pa2,&ta);
                if ((ta == tb) &&
                    ((si = compara (pb1,pb2,&pa1,&pa2,ta)) == 1))
                    calcula (rcor,&pa1,&pa2,pb1,pb2,idv,ta,nor);
            }
        break;
    }
    return (si);


* funcion que calcula la distancia desde un punto de referencia de */
* una figura hasta el punto de referencia de una figura vecina */
alcula (vieja,p1,p2,pf1,pf2,nid,t,nor)
truct ranura  *vieja;
truct punto *p1,*p2,*pf1,*pf2;
nt t,nor,nid;

    struct punto pb;
    int i;

    pb.x = vieja->refer.x ; pb.y = vieja->refer.y;
    for (i = 0; i < 3; i++) {
        desce[i].idfig = nid ; desce[i].rtipo = t;
        desce[i].desde = vieja->idfig ; desce[i].previo = vieja;
    }
    switch (t) {
    case 0 :
        if ((pf1->x <= p1->x) && (pf2->x >= p2->x))
            dishor (vieja,&pb,p1,p2,nor);
        else
        if ((p1->x <= pf1->x) && (p2->x >= pf2->x))
            dishor (vieja,&pb,pf1,pf2,nor);
        else
        if ((pf1->x <= p1->x) && (pf2->x > p1->x) && (pf2->x <= p2->x))
            dishor (vieja,&pb,p1,pf2,nor);
        else
        if ((p1->x <= pf1->x) && (p2->x > pf1->x) && (p2->x <= pf2->x))
            dishor (vieja,&pb,pf1,p2,nor);
        break;
    case 1 :
        if ((pf1->y <= p1->y) && (pf2->y >= p2->y))
            disver (vieja,&pb,p1,p2,nor);
        else
        if ((p1->y <= pf1->y) && (p2->y >= pf2->y))
            disver (vieja,&pb,pf1,pf2,nor);
        else
        if ((pf1->y <= p1->y) && (pf2->y > p1->y) && (pf2->y <= p2->y))
            disver (vieja,&pb,p1,pf2,nor);
        else
        if ((p1->y <= pf1->y) && (p2->y > pf1->y) && (p2->y <= pf2->y))
            disver (vieja,&pb,pf1,p2,nor);
        break;
    case 2 :
        dis45 (vieja,&pb,p1,p2,nor);
```

```
        break;
    case 3 :
        dis135 (vieja,&pb,p1,p2,nor);
        break;
    }
```

```
/* funcion auxiliar a calcula que considera ranuras horizontales */
/* desde un origen interior */
dishor (old,pba,p1,p2,nor)
struct ranura *old;
struct punto *pba,*p1,*p2;
int nor;

    int d;

    if ((pba->x >= p1->x) && (pba->x <= p2->x)) {
        desce[0].refer.x = pba->x;
        desce[0].refer.y = desce[1].refer.y = desce[2].refer.y = p1->y;
        d = abs (p1->y - pba->y);
        if (d == 0) {
            desce[0].dista = old->dista;
            desce[1].dista = desce[2].dista = -1;
        }
        else {
            desce[0].dista = old->dista + d;
            desce[1].dista = desce[2].dista = old->dista + d;
            switch (nor) {
            case 0 :
                desce[1].dista = desce[2].dista = -1;
                break;
            case 1 : {
                if (pba->x - d > p1->x)
                    desce[1].refer.x = pba->x - d;
                else
                if (pba->x - d <= p1->x)
                    desce[1].refer.x = p1->x;
                else
                    desce[1].dista = -1;
                if (pba->x + d < p2->x)
                    desce[2].refer.x = pba->x + d;
                else
                if (pba->x + d >= p2->x)
                    desce[2].refer.x = p2->x;
                else
                    desce[2].dista = -1;
                } break;
            }
        }
    }
    else
    if (pba->x < p1->x) {
        desce[2].dista = -1;
        pundih1 (old,pba,p1,p2,nor);
    }
```

```
   else
   if (pba->x > p2->x) {
       desce[2].dista = -1;
       pundih2 (old,pba,p1,p2,nor);
   }


* funcion auxiliar a calcula que considera ranuras verticales */
* desde un origen interior */
isver (old,pba,p1,p2,nor)
truct ranura *old;
truct punto *pba,*p1,*p2;
nt nor;

   int d;

   if ((pba->y >= p1->y) && (pba->y <= p2->y)) {
       desce[0].refer.x = desce[1].refer.x = desce[2].refer.x = p1->x;
       desce[0].refer.y = pba->y;
       d = abs (p1->x - pba->x);
       if (d == 0) {
           desce[0].dista = old->dista;
           desce[1].dista = desce[2].dista = -1;
       }
       else {
           desce[0].dista = old->dista + d;
           desce[1].dista = desce[2].dista = old->dista + d;
           switch (nor) {
           case 0 :
               desce[1].dista = desce[2].dista = -1;
               break;
           case 1 : {
               if (pba->y - d > p1->y)
                   desce[1].refer.y = pba->y - d;
               else
               if (pba->y - d <= p1->y)
                   desce[1].refer.y = p1->y;
               else
                   desce[1].dista = -1;
               if (pba->y + d < p2->y)
                   desce[2].refer.y = pba->y + d;
               else
               if (pba->y + d >= p2->y)
                   desce[2].refer.y = p2->y;
               else
                   desce[2].dista = -1;
               } break;
           }
       }
   }
   else
   if (pba->y < p1->y) {
       desce[2].dista = -1;
       pundiv1 (old,pba,p1,p2,nor);
```

```
    }
    else
    if (pba->y > p2->y) {
        desce[2].dista = -1;
        pundiv2 (old,pba,p1,p2,nor);
    }


  funcion que calcula la nueva distancia hacia una ranura */
  oblicua a 45 grados */
ls45 (vieja,pb,p1,p2,nor)
truct ranura *vieja;
truct punto *pb,*p1,*p2;
nt nor;

    int x;

    if ((pb->x < p1->x) && (pb->y - pb->x + p1->x - p1->y <= 0))
        pundis (vieja,pb,p1,nor);
    else
    if ((pb->x < p1->x) && (pb->y - pb->x + p1->x - p1->y > 0)) {
        x = abs ((pb->y + p1->x - p1->y - pb->x) / 2);
        if ((espar (x)) == 0)
            x++;
        if ((x + pb->x >= p1->x) && (x + pb->x <= p2->x))
            cal145 (vieja,pb,x,nor);
        else
            pundis (vieja,pb,p1,nor);
    }
    else
    if ((pb->x >= p1->x) && (pb->x <= p2->x) &&
        (pb->y - pb->x + p1->x - p1->y > 0)) {
        x = abs ((pb->y + p1->x - p1->y - pb->x) / 2);
        if ((espar (x)) == 0)
            x++;
        if ((x + pb->x >= p1->x) && (x + pb->x <= p2->x))
            cal145 (vieja,pb,x,nor);
        else
            pundis (vieja,pb,p2,nor);
    }
    else
    if ((pb->x > p2->x) && (pb->y - pb->x + p1->x - p1->y >= 0))
        pundis (vieja,pb,p2,nor);
    else
    if ((pb->x > p2->x) && (pb->y - pb->x + p1->x - p1->y < 0)) {
        x = abs ((pb->x - pb->y - p1->x + p1->y) / 2);
        if ((espar (x)) == 0)
            x++;
        if ((pb->x - x <= p2->x) && (pb->x - x >= p1->x))
            cal245 (vieja,pb,x,nor);
        else
            pundis (vieja,pb,p2,nor);
    }
    else
    if ((pb->x >= p1->x) && (pb->x <= p2->x) &&
```

```
        (pb->y - pb->x + p1->x - p1->y < 0)) {
        x = abs ((pb->x - pb->y - p1->x + p1->y) / 2);
        if ((espar (x)) == 0)
            x++;
        if ((pb->x - x <= p2->x) && (pb->x - x >= p1->x))
            cal245 (vieja,pb,p1,p2,nor);
        else
            pundis (vieja,pb,p1,nor);
    }


* funcion que calcula la nueva distancia hacia una ranura */
* oblicua a 135 grados */
s135 (vieja,pb,p1,p2,nor)
ruct ranura *vieja;
ruct punto *pb,*p1,*p2;
t nor;

    int x;

    if ((pb->x > p1->x) && (pb->y + pb->x - p1->x - p1->y <= 0))
        pundis (vieja,pb,p1,nor);
    else
    if ((pb->x > p1->x) && (pb->y + pb->x - p1->x - p1->y > 0)) {
        x = abs ((pb->x + pb->y - p1->x - p1->y) / 2);
        if ((espar (x)) == 0)
            x++;
        if ((pb->x - x <= p1->x) && (pb->x - x >= p2->x))
            cal2135 (vieja,pb,x,nor);
        else
            pundis (vieja,pb,p1,nor);
    }
    else
    if ((pb->x <= p1->x) && (pb->x >= p2->x) &&
        (pb->y + pb->x - p1->x - p1->y > 0)) {
        x = abs ((pb->x + pb->y - p1->x - p1->y) / 2);
        if ((espar (x)) == 0)
            x++;
        if ((pb->x - x <= p1->x) && (pb->x - x >= p2->x))
            cal2135 (vieja,pb,x,nor);
        else
            pundis (vieja,pb,p2,nor);
    }
    else
    if ((pb->x < p2->x) && (pb->y + pb->x - p1->x - p1->y >= 0))
        pundis (vieja,pb,p2,nor);
    else
    if ((pb->x < p2->x) && (pb->y + pb->x - p1->x - p1->y < 0)) {
        x = abs ((- pb->y + p1->x + p1->y - pb->x) / 2);
        if ((espar (x)) == 0)
            x++;
        if ((pb->x + x >= p2->x) && (pb->x + x <= p1->x))
            cal1135 (vieja,pb,x,nor);
        else
            pundis (vieja,pb,p2,nor);
```

```
    }
    else
    if ((pb->x <= p1->x) && (pb->x >= p2->x) &&
        (pb->y + pb->x - p1->x - p1->y < 0)) {
        x = abs ((- pb->y - p1->x + p1->y - pb->x) / 2);
        if ((espar (x)) == 0)
            x++;
        if ((pb->x + x >= p2->x) && (pb->x + x <= p1->x))
            call135 (vieja,pb,x,nor);
        else
            pundis (vieja,pb,p1,nor);
    }


* funcion auxiliar a calcula para obtener el punto de referencia */
* y la distancia acumulada */
undis (old,pbas,pv,norma)
truct ranura *old;
truct punto *pbas,*pv;
nt norma;

    int dx,dy;

    desce[0].refer.x = pv->x ; desce[0].refer.y = pv->y;
    desce[1].dista = desce[2].dista = -1;
    dx = abs (pv->x - pbas->x) ; dy = abs (pv->y - pbas->y);
    switch (norma) {
    case 0 :
        desce[0].dista = old->dista + dx + dy;
        break;
    case 1 :
        desce[0].dista = old->dista + max (dx,dy);
        break;
    }


* funcion auxiliar a calcula para obtener el punto de referencia */
* y la distancia acumulada, ranuras horizontales (caso 1) */
undih1 (old,pbas,pv1,pv2,norma)
truct ranura  *old;
truct punto *pbas,*pv1,*pv2;
nt norma;

    int dx,dy;

    desce[0].refer.x = pv1->x ;
    desce[0].refer.y = desce[1].refer.y = pv1->y;
    dx = abs (pv1->x - pbas->x) ; dy = abs (pv1->y - pbas->y);
    switch (norma) {
    case 0 : {
        desce[0].dista = old->dista + dx + dy;
        desce[1].dista = -1;
        } break;
    case 1 : {
        desce[0].dista = desce[1].dista = old->dista + max (dx,dy);
```

```
        if (dx >= dy)
            desce[1].dista = -1;
        else
        if ((pbas->x + dy > pv1->x) && (pbas->x + dy < pv2->x))
            desce[1].refer.x = pbas->x + dy;
        else
        if ((pbas->x + dy > pv1->x) && (pbas->x + dy >= pv2->x))
            desce[1].refer.x = pv2->x;
        } break;
    }


* funcion auxiliar a calcula para obtener el punto de referencia */
* y la distancia acumulada, ranuras horizontales (caso 2) */
undih2 (old,pbas,pv1,pv2,norma)
truct ranura  *old;
truct punto *pbas,*pv1,*pv2;
nt norma;

    int dx,dy;

    desce[0].refer.x = pv2->x ;
    desce[0].refer.y = desce[1].refer.y = pv1->y;
    dx = abs (pv2->x - pbas->x) ; dy = abs (pv2->y - pbas->y);
    switch (norma) {
    case 0 : {
        desce[0].dista = old->dista + dx + dy;
        desce[1].dista = -1;
        } break;
    case 1 : {
        desce[0].dista = desce[1].dista = old->dista + max (dx,dy);
        if (dx >= dy)
            desce[1].dista = -1;
        else
        if ((pbas->x - dy > pv1->x) && (pbas->x - dy < pv2->x))
            desce[1].refer.x = pbas->x - dy;
        else
        if ((pbas->x - dy <= pv1->x) && (pbas->x - dy < pv2->x))
            desce[1].refer.x = pv1->x;
        } break;
    }
}

/* funcion auxiliar a calcula para obtener el punto de referencia */
* y la distancia acumulada, ranuras verticales (caso 1) */
undivi (old,pbas,pv1,pv2,norma)
truct ranura  *old;
truct punto *pbas,*pv1,*pv2;
nt norma;

    int dx,dy;

    desce[0].refer.y = pv1->y ;
    desce[0].refer.x = desce[1].refer.x = pv1->x;
    dx = abs (pv1->x - pbas->x) ; dy = abs (pv1->y - pbas->y);
```

```
    switch (norma) {
    case 0 : {
        desce[0].dista = old->dista + dx + dy;
        desce[1].dista = -1;
        } break;
    case 1 : {
        desce[0].dista = desce[1].dista = old->dista + max (dx,dy);
        if (dy >= dx)
            desce[1].dista = -1;
        else
        if ((pbas->y + dx > pv1->y) && (pbas->y + dx < pv2->y))
            desce[1].refer.y = pbas->y + dx;
        else
        if ((pbas->y + dx > pv1->y) && (pbas->y + dx >= pv2->y))
            desce[1].refer.y = pv2->y;
        } break;
    }


* funcion auxiliar a calcula para obtener el punto de referencia */
* y la distancia acumulada, ranuras verticales (caso 2) */
undiv2 (old,pbas,pv1,pv2,norma)
truct ranura  *old;
truct punto *pbas,*pv1,*pv2;
nt norma;

    int dx,dy;

    desce[0].refer.y = pv2->y ;
    desce[0].refer.x = desce[1].refer.x = pv1->x;
    dx = abs (pv2->x - pbas->x) ; dy = abs (pv2->y - pbas->y);
    switch (norma) {
    case 0 : {
        desce[0].dista = old->dista + dx + dy;
        desce[1].dista = -1;
        } break;
    case 1 : {
        desce[0].dista = desce[1].dista = old->dista + max (dx,dy);
        if (dy >= dx)
            desce[1].dista = -1;
        else
        if ((pbas->y - dx > pv1->y) && (pbas->y - dx < pv2->y))
            desce[1].refer.y = pbas->y - dx;
        else
        if ((pbas->y - dx <= pv1->y) && (pbas->y - dx < pv2->y))
            desce[1].refer.y = pv1->y;
        } break;
    }


* funcion auxiliar a calcula para el calculo de la nueva ranura */
* si el lado es oblicuo a 45 grados (caso 1) */
al145 (old,pbas,x,norma)
truct ranura *old;
truct punto *pbas;
```

```
nt norma,x;

   desce[0].refer.x = pbas->x + x ; desce[0].refer.y = pbas->y - x;
   desce[0].dista = distanc (old,norma,x);
   desce[1].dista = desce[2].dista = -1;


* funcion auxiliar a calcula para el calculo de la nueva ranura */
* si el lado es oblicuo a 45 grados (caso 2) */
l1245 (old,pbas,x,norma)
truct ranura *old;
truct punto *pbas;
nt norma,x;

   desce[0].refer.x = pbas->x - x ; desce[0].refer.y = pbas->y + x;
   desce[0].dista = distanc (old,norma,x);
   desce[1].dista = desce[2].dista = -1;


* funcion auxiliar a calcula para el calculo de la nueva ranura */
* si el lado es oblicuo a 135 grados (caso 1) */
al1135 (old,pbas,pnu,x,norma)
truct ranura *old;
truct punto *pbas;
nt norma,x;

   desce[0].refer.x = pbas->x + x ; desce[0].refer.y = pbas->y + x;
   desce[0].dista = distanc (old,norma,x);
   desce[1].dista = desce[2].dista = -1;


* funcion auxiliar a calcula para el calculo de la nueva ranura */
* si el lado es oblicuo a 135 grados (caso 2) */
al2135 (old,pbas,pnu,x,norma)
truct ranura *old;
truct punto *pbas;
nt norma,x;

   desce[0].refer.x = pbas->x - x ; desce[0].refer.y = pbas->y - x;
   desce[0].dista = distanc (old,norma,x);
   desce[1].dista = desce[2].dista = -1;


* funcion que calcula la nueva distancia de aacuerdo a la norma de medida */
istanc (ante,nor,delta)
truct ranura *ante;
nt nor,delta;

   int dis;

   switch (nor) {
   case 0 :
       dis = ante->dista + delta + delta;
       break;
   case 1 :
```

```
        dis = ante->dista + delta;
        break;
    }
    return (dis);
}

/* funcion que muestra la propagacion de ranuras */
escribe (h)
struct ranura *h;
{
printf ("%4d%9d%10d%12d%12d\n",h->desde,h->idfig,h->refer.x,h->refer.y,n->dis
}

/* funcion que construye la cola de prioridades con las distancias */
/* acumuladas desde el punto inicial hasta una figura de la grafica */
struct queue *inserta (lsq,wdg)
struct queue *lsq;
struct ranura *wdg;
{
    struct queue *p,*q;
    struct ranura *ra,ranaux;
    char *malloc();

    p = (struct queue *) malloc(sizeof(struct queue));
    ra = (struct ranura *) malloc(sizeof(struct ranura));
    copranu (&p->info,wdg);
    copranu (ra,p->info.previo);
    p->info.previo = ra;
    p->sigui = NULL;
    if (lsq == NULL)
        lsq = p;
    else {
        q = lsq;
        while ((q->sigui != NULL) && (q->info.dista <= p->info.dista))
            q = q->sigui;
        while ((q->sigui != NULL) && (q->info.dista == p->info.dista))
            if ((yaesta (&q->info,&p->info)) == 1) {
                free ((char *) p) ; free ((char *) ra);
                return (lsq);
            }
            else
                q = q->sigui;
        if ((q != NULL) && (q->info.dista > wdg->dista)) {
            copranu (&ranaux,&q->info);
            copranu (&q->info,&p->info);
            copranu (&p->info,&ranaux);
            p->sigui = q->sigui;
        }
        q->sigui = p;
    }
    return (lsq);


' funcion que copia los campos de una ranura a otra */
>pranu (ra1,ra2)
```

```
truct ranura *ra1, *ra2;

   ra1->refer.x = ra2->refer.x ; ra1->refer.y = ra2->refer.y;
   ra1->desde = ra2->desde ; ra1->rtipo = ra2->rtipo;
   ra1->idfig = ra2->idfig ; ra1->dista = ra2->dista;
   ra1->previo = ra2->previo;


' funcion que establece la condicion de igualdad de ranuras */
qran (r1,r2)
truct ranura *r1, *r2;

   int res = 0;

   if ((r1->refer.x == r2->refer.x) && (r1->refer.y == r2->refer.y) &&
       (r1->dista == r2->dista) && (r1->desde == r2->desde) &&
       (r1->idfig == r2->idfig))
       res = 1;
   return (res);


' funcion que establece si una propagacion esta ya en la lista */
' de ranuras propagadas */
bmprue (ran.lc)
truct ranura *ran;
truct cerrado *lc;

   struct cerrado *p;
   int res = 0;

   if (lc == NULL)
       return (res);
   else {
       p = lc;
       while (p != NULL)
           if ((res = yaesta (&p->visit,ran)) == 1)
               return (res);
           else
               p = p->proxi;
   }
   return (res);


' funcion que establece la condicion para que una ranura */
 ya se encuentre en la lista de propagadas */
esta (r1,r2)
truct ranura *r1,*r2;

   int res = 0;

   if ((r1->idfig == r2->desde) && (r1->desde == r2->idfig) &&
       (r1->rtipo == r2->rtipo) && (r1->dista <= r2->dista))
       switch (r1->rtipo) {
       case 0 :
           if (r1->refer.y == r2->refer.y)
```

```
                res = 1;
            break;
        case 1 :
            if (r1->refer.x == r2->refer.x)
                res = 1;
            break;
        case 2 :
            if ((r2->refer.y - r2->refer.x + r1->refer.x - r1->refer.y) == 0)
                res = 1;
            break;
        case 3 :
            if ((r2->refer.y + r2->refer.x - r1->refer.x - r1->refer.y) == 0)
                res = 1;
            break;
        }
    else
    if ((r1->idfig == r2->idfig) && (r1->refer.x == r2->refer.x) &&
        (r1->refer.y == r2->refer.y) && (r1->dista <= r2->dista) &&
        (r1->desde == r2->desde))
        res = 1;
    return (res);


* funcion que establece si una propagacion ya esta en la cola */
ecola (lq,wdg)
truct queue *lq;
truct ranura *wdg;

    struct queue *q;
    int res = 0;

    if (lq == NULL)
        return (res);
    else {
        q = lq;
        while (q != NULL)
            if ((eqran (&q->info.wdg)) == 1) {
                res = 1;
                return (res);
            }
            else
                q = q->sigui;
    }
    return (res);


* funcion que establece si una propagacion debe reemplazar a otra */
* en la cola prioritaria (si d1 < d2) */
truct queue *substit (lq,wdg)
truct queue *lq;
truct ranura *wdg;

    struct queue *q, *anula();

    if (lq == NULL)
```

```
        return (lq);
    else {
        q = lq;
        while (q != NULL)
            if ((reempla (&q->info,wdg)) == 1) {
                lq = anula (lq,q);
                return (lq);
            }
            else
                q = q->sigui;
    }
    return (lq);


/* funcion que elimina un nodo determinado por su apuntador, */
/* en la cola prioritaria de ranuras activas */
struct queue *anula (lc,ap)
struct queue *lc,*ap;

    struct queue *p;

    if (ap == lc)
        lc = ap->sigui;
    else {
        p = lc;
        while (p->sigui != ap)
            p = p->sigui;
        p->sigui = ap->sigui;
    }
    free ((char *) ap->info.previo);
    free ((char *) ap);
    return (lc);


/* funcion que establece la condicion para que una ranura */
/* reemplace a otra en la cola prioritaria */
reempla (r1,r2)
struct ranura *r1,*r2;

    int res = 0;

    if ((r1->idfig == r2->idfig) && (r1->refer.x == r2->refer.x) &&
        (r1->refer.y == r2->refer.y) && (r1->dista > r2->dista) &&
        (r1->desde == r2->desde))
        res = 1;
    return (res);
```

3. PROGRAMA GRAFRET

ESTRUCTURAS DE DATOS

```
#define NUMVER 8     /* numero maximo de puntos adyacentes */
#define MAXRED 100    /* numero maximo de intervalos */
#define MAXOBS 100   /* numero maximo de obstaculos */

struct punto {
    int x;
    int y;
    };

struct segmento {              /* estructura para los obstaculos */
    struct punto inicial;
    struct punto final;
    int tipo;
    };

struct elstack {               /* estructura para los elementos de stacks */
    struct punto orig;
    struct elstack *previo;
    };

struct pila {                  /* estructura para stacks */
    struct elstack infor;
    struct pila *next;
    };

struct stack {                 /* estructura de pila para la ruta */
    struct punto conten;
    struct stack *sigui;
    };
```

PROGRAMA PRINCIPAL

```
#include "include\time.h"
#include "include\stdio.h"
#include "grestru.h"

extern char *malloc();

struct punto vecino [NUMVER];
int xmin,xmax,ymin,ymax;
struct segmento obst[MAXOBS];
int plano [MAXRED][MAXRED];

/* programa para calcular la ruta de longitud minima entre dos puntos */
/* en el plano, en presencia de obstaculos, cuando el plano se divide */
/* en intervalos iguales en cada direccion (grafica reticular) */
main()

    FILE *fps, *fr, *fopen();
    struct punto pi,pf,pu;
    struct elstack elem,antec,inicial,*ap;
    struct pila *stack1 = NULL, *stack2 = NULL, *stack3 = NULL;
    struct pila *push1(), *push(), *pop();
    struct stack *ruta = NULL, *empuja(), *saca();
    int desvio = 0,np = 0,nr = 0,dist,d1,d2,ca,nb,i,j,k,fclose();
    long t;

    inicio() ; datgen();
    fps = fopen ("obstac.dat","r");
    i = 0;
    while ((ca = leeobs (&obst[i],fps)) != EOF) {
        switch (obst[i].tipo) {
        case 0 :
            for (j = obst[i].inicial.x; j <= obst[i].final.x; j++)
                plano [ymax - obst[i].inicial.y][j] = -1;
            break;
        case 1 :
            for (j = obst[i].inicial.y; j <= obst[i].final.y; j++)
                plano [ymax - j][obst[i].inicial.x] = -1;
            break;
        case 2 : {
            k = ymax - obst[i].inicial.y;
            for (j = obst[i].inicial.x; j <= obst[i].final.x; j++)
                plano [k--][j] = -1;
            } break;
        case 3 : {
            k = ymax - obst[i].inicial.y;
            for (j = obst[i].inicial.x; j >= obst[i].final.x; j--)
                plano [k--][j] = -1;
            } break;
        }
        i++;
    }
    fclose (fps);
    nb = --i;
```

```
fr = fopen ("trayec.dat","w");
printf ("\nDETERMINACION DE LA RUTA DE LONGITUD MINIMA ENTRE\n");
printf ("DOS PUNTOS DEL PLANO EN PRESENCIA DE OBSTACULOS\n");
printf ("\nALGORITMO DE GRAFICA RETICULAR (S.Futagami y otros)\n");
fprintf (fr,"\nDETERMINACION DE LA RUTA DE LONGITUD MINIMA ENTRE\n");
fprintf (fr,"DOS PUNTOS DEL PLANO EN PRESENCIA DE OBSTACULOS\n");
fprintf (fr,"\nALGORITMO DE GRAFICA RETICULAR (S.Futagami y otros)\n");
printf ("\nCOORDENADAS DE LOS PUNTOS INICIAL Y FINAL\n");
printf ("\nPunto inicial\n");
printf ("Abscisa = ") ; scanf ("%d",&pi.x);
printf ("Ordenada = ") ; scanf ("%d",&pi.y);
printf ("\nPunto final\n");
printf ("Abscisa = ") ; scanf ("%d",&pf.x);
printf ("Ordenada = ") ; scanf ("%d",&pf.y);
fprintf (fr,"\nCOORDENADAS DE LOS PUNTOS INICIAL Y FINAL\n");
fprintf (fr,"\nPunto inicial\n");
fprintf (fr,"Abscisa = ") ; fprintf (fr,"%d\n",pi.x);
fprintf (fr,"Ordenada = ") ; fprintf (fr,"%d\n",pi.y);
fprintf (fr,"\nPunto final\n");
fprintf (fr,"Abscisa = ") ; fprintf (fr,"%d\n",pf.x);
fprintf (fr,"Ordenada = ") ; fprintf (fr,"%d\n",pf.y);
time (&t);
printf ("\nLa hora actual es %s\n",ctime (&t));
fprintf (fr,"\nLa hora actual es %s\n",ctime (&t));
inicial.orig.x = pi.x ; inicial.orig.y = pi.y;
inicial.previo = NULL;
copelst (&antec,&inicial);
pu.x = pi.x ; pu.y = pi.y;
while ((pu.x != pf.x) || (pu.y != pf.y)) {
    plano [ymax - pu.y][pu.x] = 1 ; nr++;
    calcula (&pu,&pf,nb);
    d1 = pf.y - pf.x + pu.x - pu.y;
    d2 = pf.y + pf.x - pu.y - pu.x;
    elem.previo = &antec;
    if (((d1 == 0) && (d2 > 0)) || ((d1 > 0) && (d2 == 0)) ||
        ((d1 == 0) && (d2 < 0)) || ((d1 < 0) && (d2 == 0)))
        for (i = 0; i < NUMVER; i++)
            if (vecino[i].x != -1) {
                elem.orig.x = vecino[i].x ; elem.orig.y = vecino[i].y;
                switch (i) {
                case 0 :
                    stack1 = push (stack1,&elem);
                    break;
                case 1 : case 2 :
                    stack2 = push (stack2,&elem);
                    break;
                case 3 : case 4 : case 5 : case 6 : case 7 :
                    stack3 = push (stack3,&elem);
                    break;
                }
            }
            else
                continue;
    else
    if (((d1 < 0) && (d2 > 0)) || ((d1 > 0) && (d2 > 0)) ||
```

```
    ((d1 > 0) && (d2 < 0)) || ((d1 < 0) && (d2 < 0)))
    for (i = 0; i < NUMVER; i++)
        if (vecino[i].x != -1) {
            elem.orig.x = vecino[i].x ; elem.orig.y = vecino[i].y;
            switch (i) {
            case 0 : case 1 : case 2 :
                stack1 = push (stack1,&elem);
                break;
            case 3 : case 4 :
                stack2 = push (stack2,&elem);
                break;
            case 5 : case 6 : case 7 :
                stack3 = push (stack3,&elem);
                break;
            }
        }
        else
            continue;
do {
    if (stack1 != NULL)
        stack1 = pop (stack1,&elem);
    else
        if ((stack2 == NULL) && (stack3 == NULL)) {
            printf ("\nNo existe ruta entre los puntos :\n");
            printf ("\nInicial :%5d%5d\n",pi.x,pi.y);
            printf ("Final :%5d%5d\n",pf.x,pf.y);
            goto falla;
        }
        else {
            if ((stack2 != NULL) && (stack3 != NULL)) {
                desvio++;
                do {
                    stack2 = pop (stack2,&elem);
                    stack1 = push1 (stack1,&elem);
                    } while (stack2 != NULL);
                do {
                    stack3 = pop (stack3,&elem);
                    stack2 = push1 (stack2,&elem);
                    } while (stack3 != NULL);
            }
            else
            if (stack2 == NULL) {
                desvio++;
                do {
                    stack3 = pop (stack3,&elem);
                    stack2 = push1 (stack2,&elem);
                    } while (stack3 != NULL);
                do {
                    stack2 = pop (stack2,&elem);
                    stack1 = push1 (stack1,&elem);
                    } while (stack2 != NULL);
            }
            else
            if (stack3 == NULL) {
                desvio++;
```

```
                    do {
                        stack2 = pop (stack2,&elem);
                        stack1 = push1 (stack1,&elem);
                        } while (stack2 != NULL);
                }
            }
        } while (plano [ymax-elem.orig.y][elem.orig.x] != 0);
    pu.x = elem.orig.x ; pu.y = elem.orig.y;
    copelst (&antec,&elem);
}
time (&t);
printf ("\nLa hora actual es %s\n",ctime (&t));
fprintf (fr,"\nLa hora actual es %s\n",ctime (&t));
ap = &antec;
do {
    pu.x = ap->orig.x ; pu.y = ap->orig.y;
    ruta = empuja (ruta,&pu);
    ap = ap->previo;
    } while (ap->previo != NULL);
ruta = empuja (ruta,&pi);
printf ("\n*** FIN DEL PROCESO DE ENRUTAMIENTO ***\n");
printf ("\nNumero de puntos visitados = %d\n",nr);
printf ("\nRUTA ENTRE EL PUNTO INICIAL Y EL PUNTO FINAL\n");
printf ("\n      PUNTO      ABSCISA     ORDENADA\n\n") ;
fprintf (fr,"\n*** FIN DEL PROCESO DE ENRUTAMIENTO ***\n");
fprintf (fr,"\nNumero de puntos visitados = %d\n",nr);
fprintf (fr,"\nRUTA ENTRE EL PUNTO INICIAL Y EL PUNTO FINAL\n");
fprintf (fr,"\n      PUNTO      ABSCISA     ORDENADA\n\n") ;
do {
    ruta = saca (ruta,&pu);
    printf ("%10d%11d%12d\n",++np,pu.x,pu.y);
    fprintf (fr,"%10d%11d%12d\n",np,pu.x,pu.y);
    } while (ruta != NULL);
dist = max (abs (pf.x - pi.x),abs (pf.y - pi.y)) + desvio;
printf ("\nLa distancia entre los puntos inicial y final ");
printf ("medida con norma Loo es : %d\n",dist);
fprintf (fr,"\nLa distancia entre los puntos inicial y final ");
fprintf (fr,"medida con norma Loo es : %d\n",dist);
fclose (fr);
falla :
```

```
/* funcion que inicializa la matriz del plano en cero */
inicio()
{
    int i,j;

    for (i = 0; i < MAXRED; i++)
        for (j = 0; j < MAXRED; j++)
            plano [i][j] = 0;
}

/* funcion que ingresa los limites del plano de estudio */
datgen ()
{
    printf ("\nLIMITES DEL PLANO DE ESTUDIO\n");
    printf ("\nAbscisa izquierda = ");
    scanf ("%d",&xmin);
    printf ("Abscisa derecha = ");
    scanf ("%d",&xmax);
    printf ("Ordenada inferior = ");
    scanf ("%d",&ymin);
    printf ("Ordenada superior = ");
    scanf ("%d",&ymax);
}

/* funcion que lee el arreglo de obstaculos */
leeobs (ob,ar)
struct segmento *ob;
FILE *ar;
{
    int c;

    c = fscanf (ar,"%5d%5d",&ob->inicial.x,&ob->inicial.y);
    if (c == EOF)
        return (c);
    else {
        fscanf (ar,"%5d%5d",&ob->final.x,&ob->final.y);
        fscanf (ar,"%5d\n",&ob->tipo);
    }
    return (c);
}

/* funcion que calcula los puntos adyacentes a un punto dado */
/* y los coloca en los diferentes tipos de stack */
calcula (pu,pt,nob)
struct punto *pu,*pt;
int nob;
{
    int i,d1,d2;

    d1 = pt->y - pt->x + pu->x - pu->y;
    d2 = pt->y + pt->x - pu->y - pu->x;
    if ((d1 == 0) && (d2 > 0)) {
        vecino[0].x = vecino[2].x = vecino[7].x = pu->x + 1;
        vecino[0].y = vecino[1].y = vecino[3].y = pu->y + 1;
        vecino[1].x = vecino[6].x = pu->x;
```

```
        vecino[2].y = vecino[4].y = pu->y;
        vecino[3].x = vecino[4].x = vecino[5].x = pu->x - 1;
        vecino[5].y = vecino[6].y = vecino[7].y = pu->y - 1;
    }
    else
    if ((d1 > 0) && (d2 == 0)) {
        vecino[0].x = vecino[1].x = vecino[3].x = pu->x - 1;
        vecino[0].y = vecino[2].y = vecino[7].y = pu->y + 1;
        vecino[1].y = vecino[6].y = pu->y;
        vecino[2].x = vecino[4].x = pu->x;
        vecino[3].y = vecino[4].y = vecino[5].y = pu->y - 1;
        vecino[5].x = vecino[6].x = vecino[7].x = pu->x + 1;
    }
    else
    if ((d1 == 0) && (d2 < 0)) {
        vecino[0].x = vecino[1].x = vecino[3].x = pu->x - 1;
        vecino[0].y = vecino[2].y = vecino[7].y = pu->y - 1;
        vecino[1].y = vecino[6].y = pu->y;
        vecino[2].x = vecino[4].x = pu->x;
        vecino[3].y = vecino[4].y = vecino[5].y = pu->y + 1;
        vecino[5].x = vecino[6].x = vecino[7].x = pu->x + 1;
    }
    else
    if ((d1 < 0) && (d2 == 0)) {
        vecino[0].x = vecino[2].x = vecino[7].x = pu->x + 1;
        vecino[0].y = vecino[1].y = vecino[3].y = pu->y - 1;
        vecino[1].x = vecino[6].x = pu->x;
        vecino[2].y = vecino[4].y = pu->y;
        vecino[3].x = vecino[4].x = vecino[5].x = pu->x - 1;
        vecino[5].y = vecino[6].y = vecino[7].y = pu->y + 1;
    }
    else
    if ((d1 < 0) && (d2 > 0)) {
        vecino[0].x = vecino[1].x = vecino[2].x = pu->x + 1;
        vecino[2].y = vecino[6].y = pu->y;
        vecino[1].y = vecino[3].y = vecino[5].y = pu->y + 1;
        vecino[0].y = vecino[4].y = vecino[7].y = pu->y - 1;
        vecino[3].x = vecino[4].x = pu->x;
        vecino[5].x = vecino[6].x = vecino[7].x = pu->x - 1;
    }
    else
    if ((d1 > 0) && (d2 > 0)) {
        vecino[2].x = vecino[6].x = pu->x;
        vecino[0].y = vecino[1].y = vecino[2].y = pu->y + 1;
        vecino[1].x = vecino[3].x = vecino[5].x = pu->x - 1;
        vecino[0].x = vecino[4].x = vecino[7].x = pu->x + 1;
        vecino[3].y = vecino[4].y = pu->y;
        vecino[5].y = vecino[6].y = vecino[7].y = pu->y - 1;
    }
    else
    if ((d1 > 0) && (d2 < 0)) {
        vecino[0].x = vecino[1].x = vecino[2].x = pu->x - 1;
        vecino[2].y = vecino[6].y = pu->y;
        vecino[1].y = vecino[3].y = vecino[5].y = pu->y + 1;
        vecino[0].y = vecino[4].y = vecino[7].y = pu->y - 1;
```

```
        vecino[3].x = vecino[4].x = pu->x;
        vecino[5].x = vecino[6].x = vecino[7].x = pu->x + 1;
    }
    else
    if ((d1 < 0) && (d2 < 0)) {
        vecino[2].x = vecino[6].x = pu->x;
        vecino[0].y = vecino[1].y = vecino[2].y = pu->y - 1;
        vecino[1].x = vecino[3].x = vecino[5].x = pu->x - 1;
        vecino[0].x = vecino[4].x = vecino[7].x = pu->x + 1;
        vecino[3].y = vecino[4].y = pu->y;
        vecino[5].y = vecino[6].y = vecino[7].y = pu->y + 1;
    }
    voblic (pu,pt,nob);
    valvec (pu,pt);
    for (i = 0; i < NUMVER; i++)
        if ((vecino[i].x != -1) &&
            (plano[ymax-vecino[i].y][vecino[i].x] == -1))
            vecino[i].x = -1;
    for (i = 0; i < NUMVER; i++)
        if ((vecino[i].x != -1) &&
            (plano[ymax-vecino[i].y][vecino[i].x] == 1))
            vecino[i].x = -1;


* funcion que valida a los puntos adyacentes de un punto dado */
* cuando el punto esta en el contorno del plano */
alvec (pu,pt)
truct punto *pu,*pt;

    int d1,d2;

    d1 = pt->y - pt->x + pu->x - pu->y;
    d2 = pt->y + pt->x - pu->y - pu->x;
    if ((d1 == 0) && (d2 > 0))
        valid03 (pu,ymin);
    else
    if ((d1 > 0) && (d2 == 0))
        valid12 (pu,ymin);
    else
    if ((d1 == 0) && (d2 < 0))
        valid12 (pu,ymax);
    else
    if ((d1 < 0) && (d2 == 0))
        valid03 (pu,ymax);
    else
    if ((d1 < 0) && (d2 > 0))
        valid46 (pu,xmin);
    else
    if ((d1 > 0) && (d2 > 0))
        valid57 (pu,ymin);
    else
    if ((d1 > 0) && (d2 < 0))
        valid46 (pu,xmax);
    else
    if ((d1 < 0) && (d2 < 0))
```

```
        valid57 (pu,ymax);
}

/* funcion que valida los puntos adyacentes en los casos 0 y 3 */
valid03 (pu,y)
struct punto *pu;
int y;
{
    if ((pu->x == xmin) && (pu->y == y)) {
        vecino[3].x = vecino[4].x = vecino[5].x =
        vecino[6].x = vecino[7].x = -1;
    }
    else
    if (pu->x == xmin) {
        vecino[3].x = vecino[4].x = vecino[5].x = -1;
    }
    else
    if (pu->y == y) {
        vecino[5].x = vecino[6].x = vecino[7].x = -1;
    }
}

/* funcion que valida los puntos adyacentes en los casos 1 y 2 */
valid12 (pu,y)
struct punto *pu;
int y;
{
    if ((pu->x == xmax) && (pu->y == y)) {
        vecino[3].x = vecino[4].x = vecino[5].x =
        vecino[6].x = vecino[7].x = -1;
    }
    else
    if (pu->x == xmax) {
        vecino[5].x = vecino[6].x = vecino[7].x = -1;
    }
    else
    if (pu->y == y) {
        vecino[3].x = vecino[4].x = vecino[5].x = -1;
    }
}

/* funcion que valida los puntos adyacentes en los casos 4 y 6 */
valid46 (pu,x)
struct punto *pu;
int x;
{
    if ((pu->x == x) && (pu->y == ymin)) {
        vecino[0].x = vecino[4].x = vecino[5].x =
        vecino[6].x = vecino[7].x = -1;
    }
    else
    if ((pu->x == x) && (pu->y == ymax)) {
        vecino[1].x = vecino[3].x = vecino[5].x =
        vecino[6].x = vecino[7].x = -1;
    }
```

```
    else
    if (pu->x == x) {
        vecino[5].x = vecino[6].x = vecino[7].x = -1;
    }
    else
    if (pu->y == ymin) {
        vecino[0].x = vecino[4].x = vecino[7].x = -1;
    }
    else
    if (pu->y == ymax) {
        vecino[1].x = vecino[3].x = vecino[5].x = -1;
    }
}

/* funcion que valida los puntos adyacentes en los casos 5 y 7 */
valid57 (pu,y)
struct punto *pu;
int y;
{
    if ((pu->x == xmin) && (pu->y == y)) {
        vecino[1].x = vecino[3].x = vecino[5].x =
        vecino[6].x = vecino[7].x = -1;
    }
    else
    if ((pu->x == xmax) && (pu->y == y)) {
        vecino[0].x = vecino[4].x = vecino[5].x =
        vecino[6].x = vecino[7].x = -1;
    }
    else
    if (pu->x == xmin) {
        vecino[1].x = vecino[3].x = vecino[5].x = -1;
    }
    else
    if (pu->y == y) {
        vecino[5].x = vecino[6].x = vecino[7].x = -1;
    }
    else
    if (pu->x == xmax) {
        vecino[0].x = vecino[4].x = vecino[7].x = -1;
    }
}

/* funcion que valida a los puntos adyacentes de un punto dado */
/* cuando el punto esta contiguo a un obstaculo oblicuo */
voblic (pu,pt,n)
struct punto *pu,*pt;
int n;
{
    int d1,d2,i;

    d1 = pt->y - pt->x + pu->x - pu->y;
    d2 = pt->y + pt->x - pu->y - pu->x;
    for (i = 0; i <= n; i++)
        switch (obst[i].tipo) {
        case 0 : case 1 :
```

```
            break;
        case 2 : case 3 :
            if ((d1 == 0) && (d2 > 0))
                vobli0 (i);
            else
            if ((d1 > 0) && (d2 == 0))
                vobli1 (i);
            else
            if ((d1 == 0) && (d2 < 0))
                vobli2 (i);
            else
            if ((d1 < 0) && (d2 == 0))
                vobli3 (i);
            else
            if ((d1 < 0) && (d2 > 0))
                vobli4 (i);
            else
            if ((d1 > 0) && (d2 > 0))
                vobli5 (i);
            else
            if ((d1 > 0) && (d2 < 0))
                vobli6 (i);
            else
            if ((d1 < 0) && (d2 < 0))
                vobli7 (i);
            break;
        }
}

/* funcion complementaria a voblic para el caso 0 */
vobli0 (io)
int io;
{
    int res;

    switch (obst[io].tipo) {
    case 2 : {
        if ((res = compara (&vecino[4],&vecino[1],&obst[io].inicial,
                    &obst[io].final,obst[io].tipo)) == 1)
            vecino[3].x = -1;
        else
        if ((res = compara (&vecino[6],&vecino[2],&obst[io].inicial,
                    &obst[io].final,obst[io].tipo)) == 1)
            vecino[7].x = -1;
        } break;
    case 3 : {
        if ((res = compara (&vecino[6],&vecino[4],&obst[io].inicial,
                    &obst[io].final,obst[io].tipo)) == 1)
            vecino[5].x = -1;
        else
        if ((res = compara (&vecino[2],&vecino[1],&obst[io].inicial,
                    &obst[io].final,obst[io].tipo)) == 1)
            vecino[0].x = -1;
        } break;
    }
```

```
/* funcion complementaria a voblic para el caso 1 */
voblil (io)
int io;
{
    int res;

    switch (obst[io].tipo) {
    case 2 : {
        if ((res = compara (&vecino[1],&vecino[2],&obst[io].inicial,
                    &obst[io].final,obst[io].tipo)) == 1)
            vecino[0].x = -1;
        else
        if ((res = compara (&vecino[4],&vecino[6],&obst[io].inicial,
                    &obst[io].final,obst[io].tipo)) == 1)
            vecino[5].x = -1;
        } break;
    case 3 : {
        if ((res = compara (&vecino[6],&vecino[2],&obst[io].inicial,
                    &obst[io].final,obst[io].tipo)) == 1)
            vecino[7].x = -1;
        else
        if ((res = compara (&vecino[4],&vecino[1],&obst[io].inicial,
                    &obst[io].final,obst[io].tipo)) == 1)
            vecino[3].x = -1;
        } break;
    }
}

/* funcion complementaria a voblic para el caso 2 */
voblil (io)
int io;
{
    int res;

    switch (obst[io].tipo) {
    case 2 : {
        if ((res = compara (&vecino[1],&vecino[4],&obst[io].inicial,
                    &obst[io].final,obst[io].tipo)) == 1)
            vecino[3].x = -1;
        else
        if ((res = compara (&vecino[2],&vecino[6],&obst[io].inicial,
                    &obst[io].final,obst[io].tipo)) == 1)
            vecino[7].x = -1;
        } break;
    case 3 : {
        if ((res = compara (&vecino[6],&vecino[4],&obst[io].inicial,
                    &obst[io].final,obst[io].tipo)) == 1)
            vecino[5].x = -1;
        else
        if ((res = compara (&vecino[2],&vecino[1],&obst[io].inicial,
                    &obst[io].final,obst[io].tipo)) == 1)
            vecino[0].x = -1;
        } break;
```

```
    }
}

/* funcion complementaria a voblic para el caso 3 */
vobli3 (io)
int io;
{
    int res;

    switch (obst[io].tipo) {
    case 2 : {
        if ((res = compara (&vecino[1],&vecino[2],&obst[io].inicial,
                    &obst[io].final,obst[io].tipo)) == 1)
            vecino[0].x = -1;
        else
        if ((res = compara (&vecino[4],&vecino[6],&obst[io].inicial,
                    &obst[io].final,obst[io].tipo)) == 1)
            vecino[5].x = -1;
        } break;
    case 3 : {
        if ((res = compara (&vecino[1],&vecino[4],&obst[io].inicial,
                    &obst[io].final,obst[io].tipo)) == 1)
            vecino[3].x = -1;
        else
        if ((res = compara (&vecino[2],&vecino[6],&obst[io].inicial,
                    &obst[io].final,obst[io].tipo)) == 1)
            vecino[7].x = -1;
        } break;
    }
}

/* funcion complementaria a voblic para el caso 4 */
vobli4 (io)
int io;
{
    int res;

    switch (obst[io].tipo) {
    case 2 : {
        if ((res = compara (&vecino[6],&vecino[3],&obst[io].inicial,
                    &obst[io].final,obst[io].tipo)) == 1)
            vecino[5].x = -1;
        else
        if ((res = compara (&vecino[4],&vecino[2],&obst[io].inicial,
                    &obst[io].final,obst[io].tipo)) == 1)
            vecino[0].x = -1;
        } break;
    case 3 : {
        if ((res = compara (&vecino[4],&vecino[6],&obst[io].inicial,
                    &obst[io].final,obst[io].tipo)) == 1)
            vecino[7].x = -1;
        else
        if ((res = compara (&vecino[2],&vecino[3],&obst[io].inicial,
                    &obst[io].final,obst[io].tipo)) == 1)
            vecino[1].x = -1;
```

```
        } break;
    }


/* funcion complementaria a voblic para el caso 5 */
/obli5 (io)                                    -!
int io;
{
    int res;

    switch (obst[io].tipo) {
    case 2 : {
        if ((res = compara (&vecino[6],&vecino[4],&obst[io].inicial,
                    &obst[io].final,obst[io].tipo)) == 1)
            vecino[7].x = -1;
        else
        if ((res = compara (&vecino[3],&vecino[2],&obst[io].inicial,
                    &obst[io].final,obst[io].tipo)) == 1)
            vecino[1].x = -1;
        } break;
    case 3 : {
        if ((res = compara (&vecino[4],&vecino[2],&obst[io].inicial,
                    &obst[io].final,obst[io].tipo)) == 1)
            vecino[0].x = -1;
        else
        if ((res = compara (&vecino[6],&vecino[3],&obst[io].inicial,
                    &obst[io].final,obst[io].tipo)) == 1)
            vecino[5].x = -1;
        } break;
    }


/* funcion complementaria a voblic para el caso 6 */
/obli6 (io)
int io;

    int res;

    switch (obst[io].tipo) {
    case 2 : {
        if ((res = compara (&vecino[4],&vecino[6],&obst[io].inicial,
                    &obst[io].final,obst[io].tipo)) == 1)
            vecino[7].x = -1;
        else
        if ((res = compara (&vecino[2],&vecino[3],&obst[io].inicial,
                    &obst[io].final,obst[io].tipo)) == 1)
            vecino[1].x = -1;
        } break;
    case 3 : {
        if ((res = compara (&vecino[4],&vecino[2],&obst[io].inicial,
                    &obst[io].final,obst[io].tipo)) == 1)
            vecino[0].x = -1;
        else
        if ((res = compara (&vecino[6],&vecino[3],&obst[io].inicial,
                    &obst[io].final,obst[io].tipo)) == 1)
```

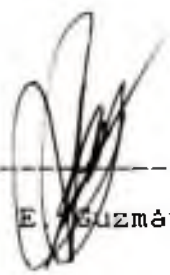
```
            vecino[5].x = -1;
        } break;
    }
}

/* funcion complementaria a voblic para el caso 7 */
voli7 (io)
int io;
{
    int res;

    switch (obst[io].tipo) {
    case 2 : {
        if ((res = compara (&vecino[3],&vecino[6],&obst[io].inicial,
                    &obst[io].final,obst[io].tipo)) == 1)
            vecino[5].x = -1;
        else
        if ((res = compara (&vecino[2],&vecino[4],&obst[io].inicial,
                    &obst[io].final,obst[io].tipo)) == 1)
            vecino[0].x = -1;
        } break;
    case 3 : {
        if ((res = compara (&vecino[4],&vecino[6],&obst[io].inicial,
                    &obst[io].final,obst[io].tipo)) == 1)
            vecino[7].x = -1;
        else
        if ((res = compara (&vecino[2],&vecino[3],&obst[io].inicial,
                    &obst[io].final,obst[io].tipo)) == 1)
            vecino[1].x = -1;
        } break;
    }
}

/* funcion que empuja un elemento en la pila de elementos */
struct pila *push (t,wdg)
struct pila *t;
struct elstack *wdg;
{
    struct pila *p;
    struct elstack *st;
    char *malloc();

    p = (struct pila *) malloc(sizeof(struct pila));
    st = (struct elstack *) malloc(sizeof(struct elstack));
    copelst (&p->infor,wdg);
    copelst (st,p->infor.previo);
    p->infor.previo = st;
    p->next = t;
    t = p;
    return (t);


* funcion que recupera un elemento del tope de la pila */
truct pila *pop (t,wdg)
truct pila *t;
```

```
struct elstack *wdg;
{
    struct pila *p;

    if (t == NULL)
        return (NULL);
    else {
        p = t;
        t = p->next;
        copelst (wdg,&p->infor);
        free ((char *) p);
        return (t);
    }
}

/* funcion que empuja un elemento en la pila de elementos */
struct pila *push1 (t,wdg)
struct pila *t;
struct elstack *wdg;
{
    struct pila *p;
    char *malloc();

    p = (struct pila *) malloc(sizeof(struct pila));
    copelst (&p->infor,wdg);
    p->next = t;
    t = p;
    return (t);
}

/* funcion que empuja un elemento en la pila de elementos */
struct stack *empuja (t,pt)
struct stack *t;
struct punto *pt;
{
    struct stack *p;
    char *malloc();

    p = (struct stack *) malloc(sizeof(struct stack));
    p->conten.x = pt->x ; p->conten.y = pt->y;
    p->sigui = t;
    t = p;
    return (t);
}

/* funcion que recupera un elemento del tope de la pila */
struct stack *saca (t,pt)
struct stack *t;
struct punto *pt;
{
    struct stack *p;

    if (t == NULL)
        return (NULL);
    else {
```

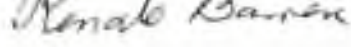
```
        p = t;
        t = p->sigui;
        pt->x = p->conten.x ; pt->y = p->conten.y;
        free ((char *) p);
        return (t);
    }

/* funcion que copia los campos de un elemento de stack */
copelst (elst1,elst2)
struct elstack *elst1,*elst2;

    elst1->orig.x = elst2->orig.x ; elst1->orig.y = elst2->orig.y;
    elst1->previo = elst2->previo;


/* funcion que compara si dos lados dados por sus vertices son comunes */
compara (ps1,ps2,pv1,pv2,tv)
struct punto *ps1,*ps2,*pv1,*pv2;
int tv;

    int da,db,res;

    if ((ps1->x != -1) && (ps2->x != -1))
        switch (tv) {
        case 0 : case 1 :
            break;
        case 2 : {
            da = ps1->y - ps1->x + pv1->x - pv1->y;
            db = ps2->y - ps2->x + pv1->x - pv1->y;
            if ((da == 0) && (db == 0) &&
                (((ps1->x >= pv1->x) && (ps2->x <= pv2->x)) ||
                ((ps1->x < pv1->x) && (ps2->x > pv2->x)) ||
                ((ps1->x < pv1->x) && (ps2->x <= pv2->x) && (ps2->x > pv1->x)) ||
                ((ps1->x >= pv1->x) && (ps1->x < pv2->x) && (ps2->x > pv2->x))))
                res = 1;
            else
                res = 0;
            }
            break;
        case 3 : {
            da = ps1->y + ps1->x - pv1->x - pv1->y;
            db = ps2->y + ps2->x - pv1->x - pv1->y;
            if ((da == 0) && (db == 0) &&
                (((ps1->y >= pv1->y) && (ps2->y <= pv2->y)) ||
                ((ps1->y < pv1->y) && (ps2->y > pv2->y)) ||
                ((ps1->y < pv1->y) && (ps2->y <= pv2->y) && (ps2->y > pv1->y)) ||
                ((ps1->y >= pv1->y) && (ps1->y < pv2->y) && (ps2->y > pv2->y))))
                res = 1;
            else
                res = 0;
            }
            break;
        }
        else
```

```
            res = 0;
    return (res);
}
```

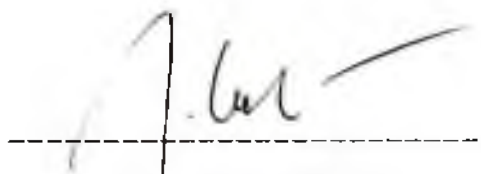
El jurado designado por la Sección de Computación del Departamento de Ingeniería Eléctrica del Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional aprobó esta tesis el 27 de Noviembre de 1987.



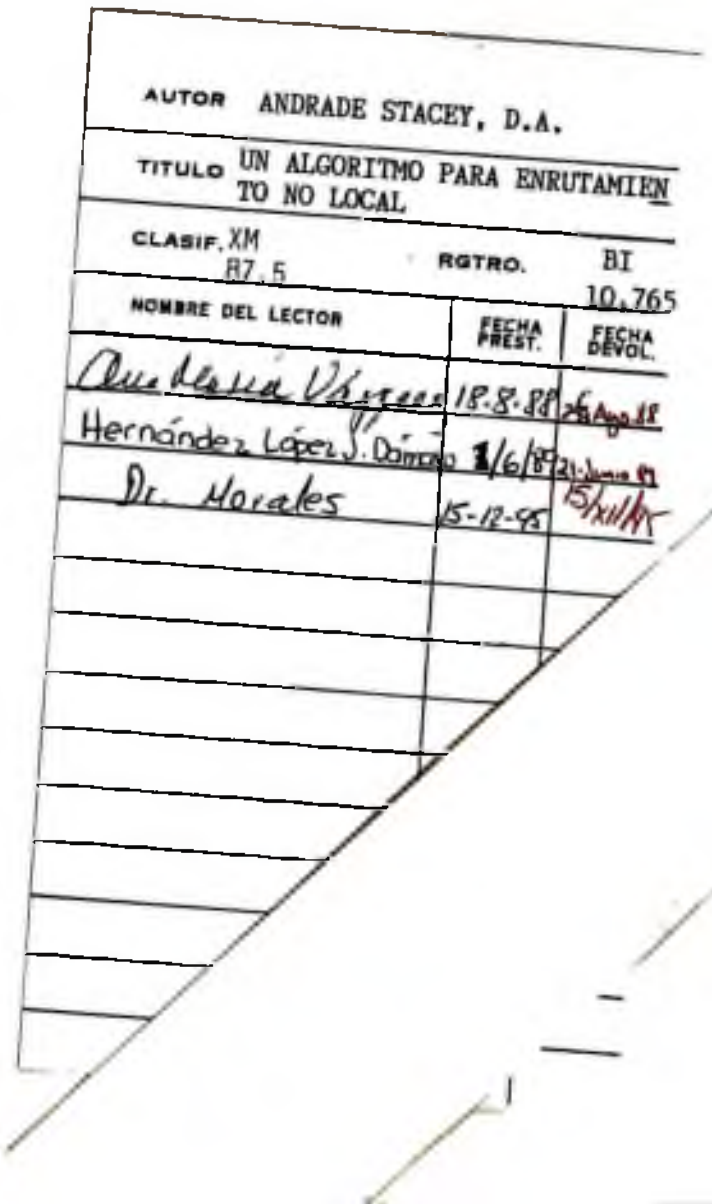
Dr. Manuel E. Guzmán Rentería



Dr. Renato Barrera Rivera



Dr. Josef Kolar



AUTOR ANDRADE STACEY, D.A.

TITULO UN ALGORITMO PARA ENRUTAMIEN TO NO LOCAL

CLASIF. XM R7.5 RGTRO. BI 10.765

NOMBRE DEL LECTOR	FECHA PREST.	FECHA DEVOL.
Ausencia Vázquez	18.8.88	26 Ago 88
Hernández López J. Dámaso	1/6/89	21 Junio 89
Dr. Morales	15-12-95	15/XII/95