FIB Facultat d'Informàtica de Barcelona

Departament d'Enginyeria de Sistemes, Automàtica i Informàtica Industrial
UNIVERSITAT POLITÈCNICA DE CATALUNYA

VISIÓ PER COMPUTADOR

Exercici 4 de Laboratori

Facultat d'Informàtica de Barcelona

Adrian Cristian Crisan

Filip Gedung Dorm

Pablo Vega Gallego

Barcelona, Octubre de 2021

Índice

Introducción

El objetivo de esta sesión es obtener una aplicación útil de visión por computador que encuentre el camino más corto entre dos puntos. Con esta aplicación se quiere discernir cual de los dos puntos es más cercano al centro. Los dos puntos a considerar están indicados en verde, azul respectivamente y el del centro está indicado en rojo, como se puede observar en la siguiente figura (fig. 1). Para encontrar el camino más corto, consideraremos que no se pueden traspasar los muros del laberinto y que el objeto que se traslada es puntual (de medida 1 píxel).

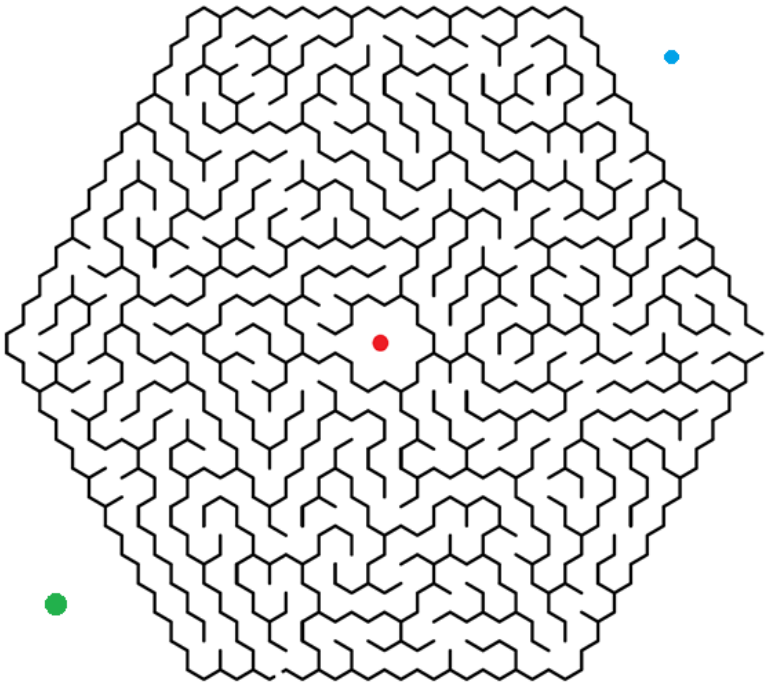

Fig. 1 Imagen del laberinto con los dos puntos a considerar

Para esta solución:

- Indicaremos la distancia (medida en píxeles) que tiene el camino más corto que une cada uno de los puntos con el centro
- Mostraremos en amarillo el camino a seguir para ir de un punto (azul o verde) hasta el centro. Al mostrar el camino no aparecerán puntos que no pertenecen al camino más corto.

Opción 1 - morfología

Leemos la imagen del problema y convertimos la imagen a escala de grises con el objetivo de encontrar los bordes más fácilmente.

```
laberinto = imread('Laberint.png');
bordes = rgb2gray(laberinto);
bordes = bordes > 60;
imshow(bordes);
```

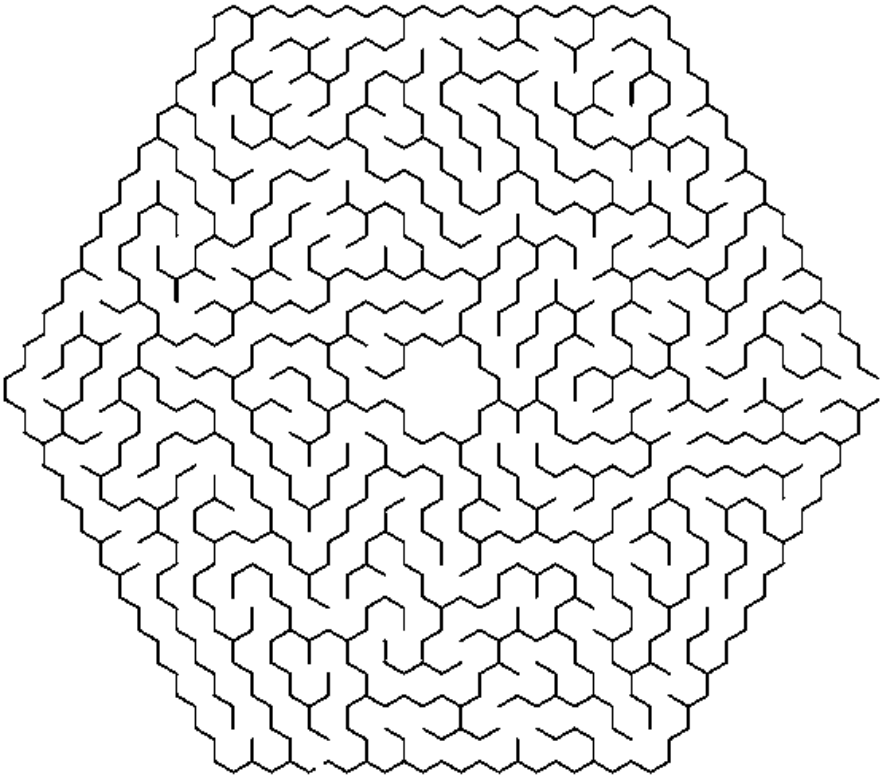

Fig. 2 Imagen del laberinto sólo con los bordes

Ahora realizamos una serie de comparaciones para poder encontrar los puntos a considerar y binarizarlas.

```
bola_roja = laberinto(:, :, 1) > 100 & laberinto(:, :, 2) < 100 & laberinto(:, :,
3) < 100;
bola_verde = laberinto(:, :, 2) > 100 & laberinto(:, :, 1) < 100 & laberinto(:, :,
3) < 100;
bola_azul = laberinto(:, :, 3) > 100 & laberinto(:, :, 2) < 200 & laberinto(:, :,
1) < 100;
imshow(bola_roja);
imshow(bola_verde);
imshow(bola_azul);
```

Fig. 3 Bola roja

Fig. 4 Bola verde

Fig. 5 Bola azul

Ahora que tenemos nuestro espacio dividido en puntos y bordes, ya podemos empezar a realizar las operaciones morfológicas para encontrar la solución óptima al laberinto presentado.

Comenzaremos creando un elemento estructural de vecindad 8, esto hará que cada vez que dilatemos la imagen aumente un píxel.

Posteriormente, utilizaremos una función “espacio_visitado” que realiza las siguientes operaciones: partimos del inicio, en este caso una de las bolas, y ponemos como objetivo la bola roja. La función consta de un bucle que va dilatando el estado inicial constantemente, al encontrarse con el borde del laberinto esta dilatación se corta imposibilitando poder saltarnos los obstáculos. Cada vez que se realiza esta iteración, aumenta una variable, que resultará ser la distancia mínima que hay desde el inicio al objetivo.

```
SE = [1, 1, 1; 1, 1, 1; 1, 1 ,1];

[distancia_verde_rojo, espacio_visitado_verde_rojo] = espacio_visitado(bola_verde,
bola_roja, bordes, SE);
[distancia_azul_rojo, espacio_visitado_azul_rojo] = espacio_visitado(bola_azul,
bola_roja, bordes, SE);

display(distancia_verde_rojo);
display(distancia_azul_rojo);
```

Las distancias resultan ser:

- Del punto verde al rojo: 1013 píxeles
- Del punto azul al rojo: 1333 píxeles

Ahora mostraremos el espacio recorrido por la dilatación desde el estado inicial al objetivo:

```
montage({espacio_visitado_verde_rojo, espacio_visitado_azul_rojo});
```

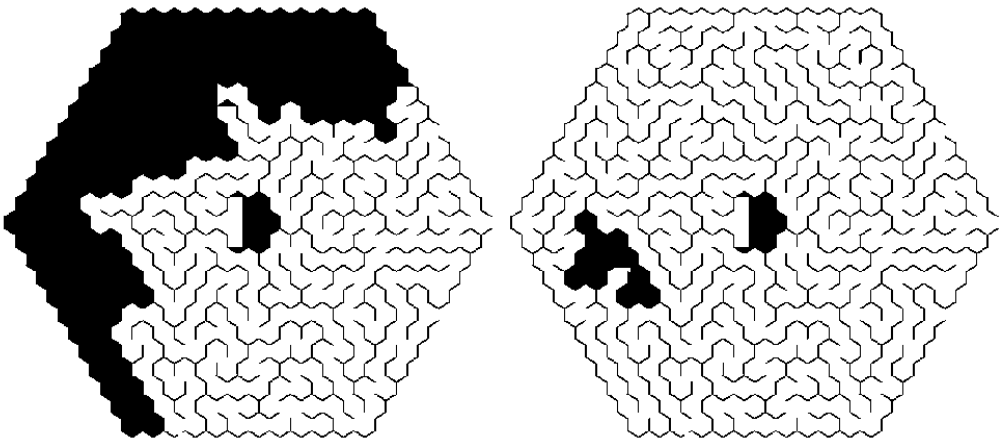

Fig. 6 Espacio visitado desde el punto verde al rojo y desde el punto azul al rojo, respectivamente

Para mostrar el camino mínimo, aprovecharemos para realizar una matriz que vaya almacenando la distancia de cada píxel al estado inicial. Si hacemos esto desde el estado inicial hacia el final y al revés, a la hora de sumar estas dos matrices, los valores que sean iguales a la distancia mínima más 1, será la zona del camino mínimo.

```
mapa_de_calor_verde_rojo = mapa_de_calor(bola_verde, bola_roja, bordes, SE);
mapa_de_calor_rojo_verde = mapa_de_calor(bola_roja, bola_verde, bordes, SE);

mapa_de_calor_azul_rojo = mapa_de_calor(bola_azul, bola_roja, bordes, SE);
mapa_de_calor_rojo_azul = mapa_de_calor(bola_roja, bola_azul, bordes, SE);

camino_verde_rojo = mapa_de_calor_verde_rojo + mapa_de_calor_rojo_verde;
camino_azul_rojo = mapa_de_calor_azul_rojo + mapa_de_calor_rojo_azul;

camino_verde_rojo = camino_verde_rojo == distancia_verde_rojo + 1;
camino_azul_rojo = camino_azul_rojo == distancia_azul_rojo + 1;

montage({camino_verde_rojo, camino_azul_rojo});
```

Fig. 7 Camino del verde al rojo y del azul al rojo, respectivamente

Reducimos este camino, para obtener el resultado con medida 1 píxel, mediante la función bwskel.

```
camino_verde_rojo = bwskel(camino_verde_rojo);
camino_azul_rojo = bwskel(camino_azul_rojo);

montage({camino_verde_rojo, camino_azul_rojo})
```

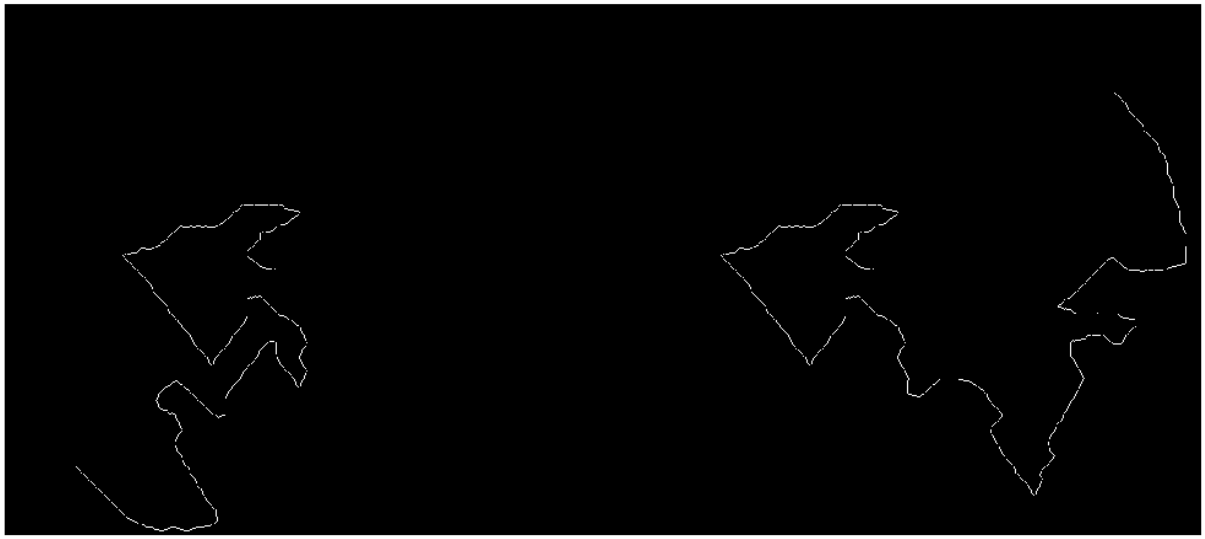

Fig. 8 Camino a tamaño 1 píxel del verde al rojo y del azul al rojo, respectivamente

Finalmente, hacemos el overlay de la imagen del camino con el original y pintamos de amarillo este camino.

```
solucion_verde = imoverlay(laberinto, camino_verde_rojo, 'yellow');
solucion_azul = imoverlay(laberinto, camino_azul_rojo, 'yellow');

montage({solucion_verde, solucion_azul});
```

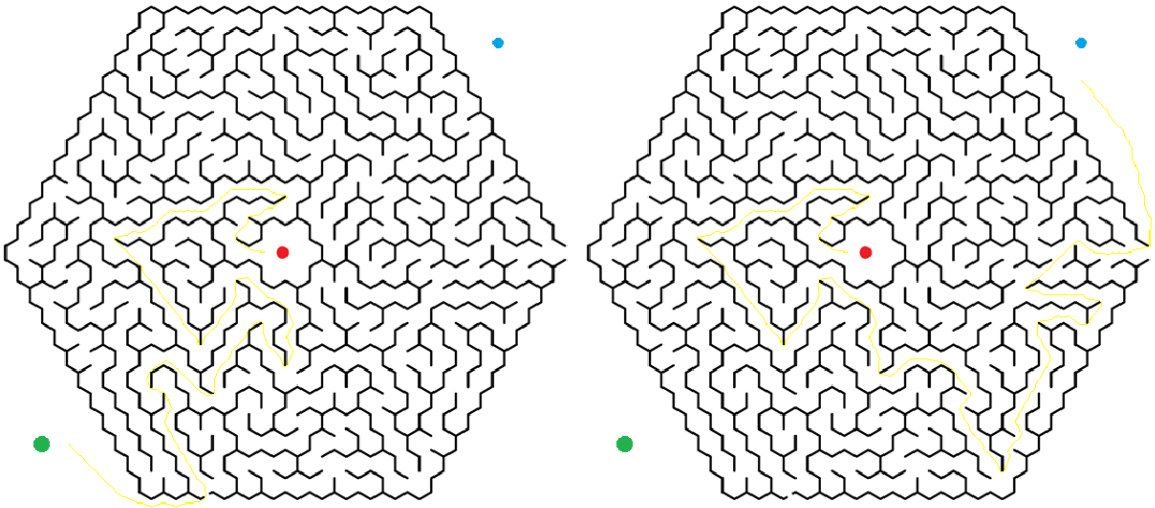

Fig. 9 Resultado final de los dos caminos: verde a rojo y azul a rojo, respectivamente

Aquí tenemos una imagen para ver este camino dilatado, para que se pueda observar mejor:

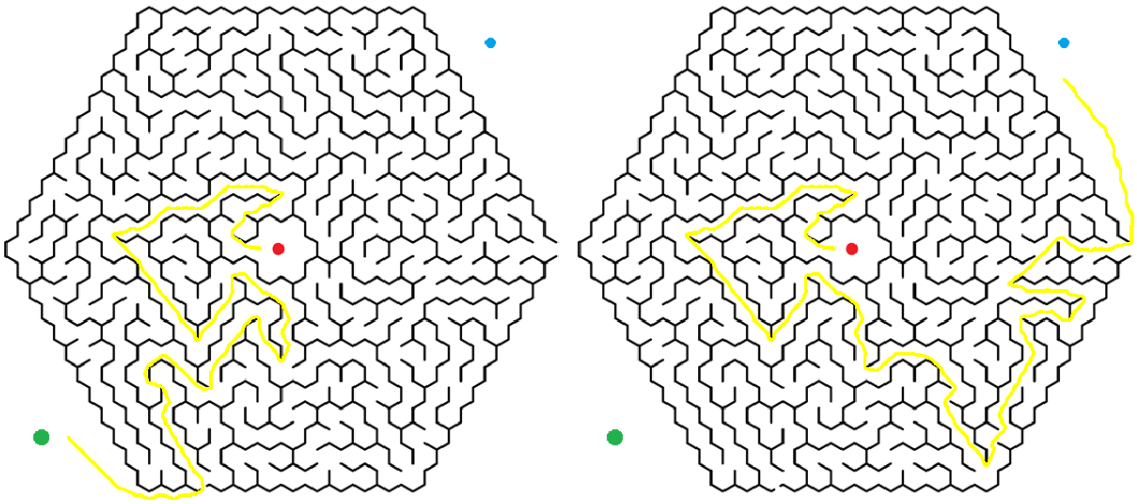

Fig. 10 Resultado con el camino dilatado

Las funciones utilizadas, para realizar este ejercicio:

```
function [distancia, visitado] = espacio_visitado(estado_inicial, estado_final,
bordes_lab, elemento_estructural)
    pixel = 1;

    while sum(sum(estado_final&estado_inicial)) == 0
        pixel = pixel + 1;
        estado_inicial = imdilate(estado_inicial, elemento_estructural);
        estado_inicial = estado_inicial&bordes_lab;
    end

    distancia = pixel;
    visitado = estado_inicial;
end


function mapa_de_grises = mapa_de_calor(estado_inicial, estado_final, bordes_lab,
elemento_estructural)
    pixel = 1;
    distancias = im2double(estado_inicial);

    while sum(sum(estado_inicial&estado_final)) == 0
        pixel = pixel + 1;
        antes = estado_inicial;
        estado_inicial = imdilate(estado_inicial, elemento_estructural);
        estado_inicial = estado_inicial&bordes_lab;
        diferencia = estado_inicial - antes;
        distancias = distancias + diferencia .* pixel;
    end

    mapa_de_grises = distancias;
end
```

Opción 2 – BFS

Leemos la imagen y binarizamos los puntos (rojo, verde y azul) para poder encontrar las coordenadas que corresponden al centro de los puntos

```
img = imread("Laberint.png");
red_point = img(:, :, 1) > 100 & img(:, :, 2) < 100 & img(:, :, 3) < 100;
green_point = img(:, :, 2) > 100 & img(:, :, 1) < 100 & img(:, :, 3) < 100;
blue_point = img(:, :, 3) > 100 & img(:, :, 2) < 200 & img(:, :, 1) < 100;
[Rmx, Rmy] = get_middle(red_point);
[Gmx, Gmy] = get_middle(green_point);
[Bmx, Bmy] = get_middle(blue_point);
```

Aplicamos el algoritmo BFS para encontrar la distancia entre los puntos y obtenemos las distancias y las imágenes con el camino más corto.

```
[dist_green_dot, img_green_dot] = BFS(img, Gmx, Gmy, Rmx, Rmy, green_point,
red_point);
[dist_blue_dot, img_blue_dot] = BFS(img, Bmx, Bmy, Rmx, Rmy, blue_point,
red_point);
dist_green_dot
dist_blue_dot
montage({img_green_dot, img_blue_dot})
```

Las distancias resultan ser:

- Del punto verde al rojo: 1013 píxeles
- Del punto azul al rojo: 1332 píxeles

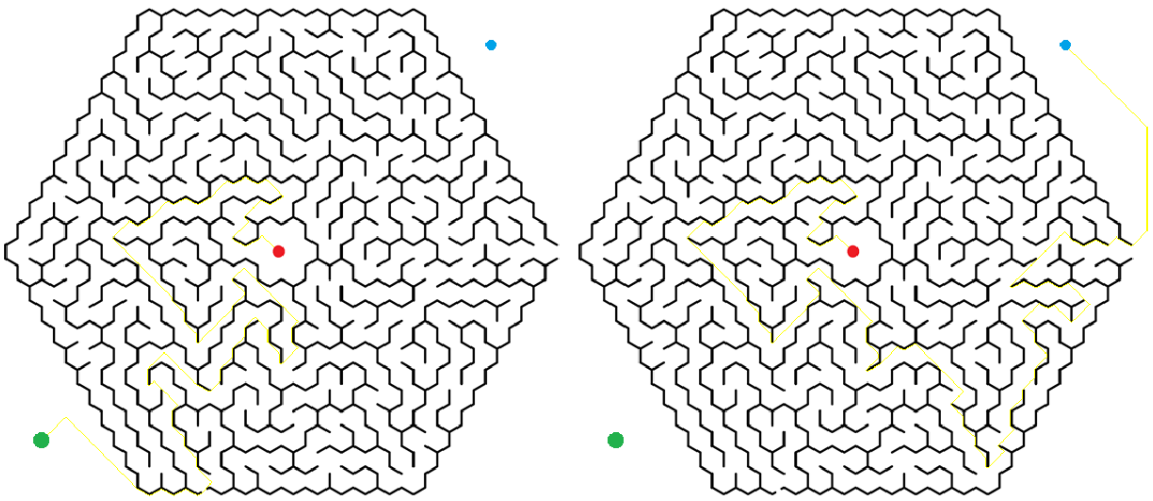

Fig. 11 Resultado final de los dos caminos: verde a rojo y azul a rojo, respectivamente

Para dilatar el camino para que sea más visible:

```
img_blue_red = img_blue_dot(:, :, 1) == 255 & img_blue_dot(:, :, 2) == 255 &
img_blue_dot(:, :, 3) == 0;
SE = [1, 1, 1; 1, 1, 1; 1, 1 ,1];
img_blue_red = imdilate(img_blue_red, SE);
img_blue_red = imoverlay(img, img_blue_red);

img_green_red = img_green_dot(:, :, 1) == 255 & img_green_dot(:, :, 2) == 255 &
img_green_dot(:, :, 3) == 0;
```

```
img_green_red = imdilate(img_green_red, SE);
img_green_red = imoverlay(img, img_green_red);

montage({img_blue_red, img_green_red})
```

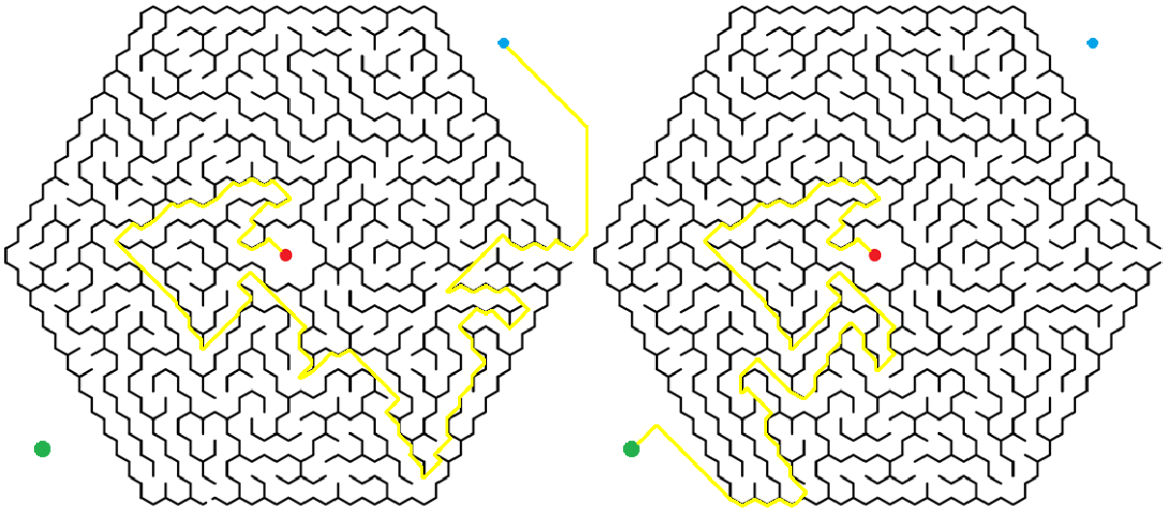

Fig. 12 Resultado con el camino dilatado

La función del BFS usado:

```
function [dist, img] = BFS(img, sourceX, sourceY, destinationX, destinationY, ini,
fin)
% Obtenemos los bordes del laberinto
bordes = rgb2gray(img);
bordes = bordes > 60;

% Matrices para conocer la posición previa
previousX(1:505, 1:570) = -1;
previousY(1:505, 1:570) = -1;
% Matriz que para conocer las distancias
distance(1:505, 1:570) = -1;
% Array que hace la función de una cola
queue(1).x = sourceX;
queue(1).y = sourceY;
distance(sourceX, sourceY) = 0;

% Posiciones adyacentes (vecinos)
moveX = [-1, 0, 1, -1, 1, -1, 0, 1];
moveY = [1, 1, 1, 0, 0, -1, -1, -1];
move.x = moveX;
move.y = moveY;

while (isempty(queue) == 0)
    x = queue(1).x;
    y = queue(1).y;
    queue = queue(2:end);

    if (x == destinationX && y == destinationY)
        dist = distance(x, y);
        break;
    end
```

```
    m = size(move.x);
    for i = 1:m(2)
        x1 = x + move.x(i);
        y1 = y + move.y(i);
        if (x1 <= 505 && 0 < x1 && y1 <= 570 && 0 < y1 && bordes(x1, y1) == 1)
            if (distance(x1, y1) == -1)
                distance(x1, y1) = distance(x, y) + 1;
                queue(end+1).x = x1;
                queue(end).y = y1;
                previousX(x1, y1) = x;
                previousY(x1, y1) = y;
            end
        end
    end
end

% Pintamos los píxeles que nos llevan por camino más corto de color amarillo
prevX = x; prevY = y;
while (prevX ~= -1)
    % Esta condicion la ponemos para que no pinte los puntos y no tenga
    % en cuente como distancia los píxeles dentro de los puntos
    if (ini(prevX, prevY) == 0 && fin(prevX, prevY) == 0)
        img(prevX, prevY, 1) = 255;
        img(prevX, prevY, 2) = 255;
        img(prevX, prevY, 3) = 0;
    else
        dist = dist - 1;
    end
    auxX = previousX(prevX, prevY);
    prevY = previousY(prevX, prevY);
    prevX = auxX;
end
end
```

Dado que Matlab no tiene tiene una estructura de colas, hemos adaptado un array para que la sustituya.

Para obtener la distancia tenemos una matriz de distancias, podríamos haber hecho también una matriz de booleanos que indique si ha sido visitado o no el píxel y posteriormente al recorrer el camino para pintarlo, contar los píxeles.

Podemos observar que hemos usado vecindad 8, aunque también podríamos adaptarlo a vecindad 4, el cual cambiaría ligeramente el resultado.

El algoritmo es un algoritmo simple de BFS que una vez encuentra las coordenadas que deseamos para de ejecutarse. También podemos ver que vamos guardando dentro del segundo bucle las posiciones previas para luego posteriormente poder recorrer el camino y pintar píxel a píxel.

La función usada para calcular el centro de los puntos para usarlos cómo coordenadas:

```
function [mx, my] = get_middle(point)
m = size(point);
mxH = -1; myH = -1; mxL = -1; myL = -1;
```

```
for i = 1:m(1)
    for j = 1:m(2)
        if (point(i, j) == 1)
            if (mxH == -1)
                mxH = i;
                mxL = i;
                myL = j;
                myH = j;
            else
                if (i < mxL)
                    mxL = i;
                elseif (i > mxH)
                    mxH = i;
                end
                if (j < myL)
                    myL = j;
                elseif (j > myH)
                    myH = j;
                end
            end
        end
    end
end

mx = int16((mxH + mxL) / 2);
my = int16((myH + myL) / 2);
end
```