Universitat Politècnica de Catalunya

Visió per Computador
Grau en Enginyeria Informàtica

EXERCICI 5 DE LABORATORI

SOLVING A MAZE WITH MORPHOLOGY

Autor:
Daniel DONATE

Professor:
Manel FRIGOLA
Q2 Curs 2020-2021

23 de març del 2021

UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Enunciat de l'exercici

En aquesta cinquena sessió de laboratori volem desenvolupar una senzilla aplicació de VC, amb la finalitat de trobar el camí més curt entre dos punts, però sota determinades condicions. En particular, volem trobar el camí més curt entre dos punts, tal que un d'ells està dins d'un laberint i l'altre està a fora, com en l'escenari que es suggereix a la *Figura 1*.

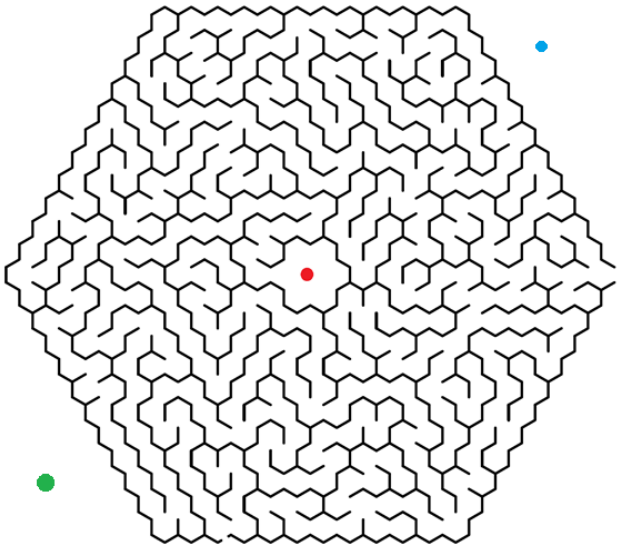

Figura 1: Imatge del laberint amb els dos punts (verd i blau) a considerar

El propòsit de l'algoritme que construirem és traçar el camí més curt des dels centres dels punts verd i blau fins al centre del punt vermell, respectivament, i després calcular quin dels dos camins és el més curt. En cercar aquest camí, lògicament, no podem traspassar els murs del laberint, sinó que hem de trobar una manera de recorre'l (fins al centre, punt vermell).
En aquest exercici presentarem, primer, les idees que hem fet servir per resoldre el problema i, després, passarem a veure la construcció de l'algoritme que finalment hem escrit.

Algoritme de camins mínims basat en una transformació de distància

L'eina principal que hem fet servir per resoldre aquest problema ha estat la transformada de distància. Recordem que la transformada de distància d'una imatge binària BW és una operació que genera una nova matriu D tal que a cada píxel de BW se li assigna un nombre en D que és la distància entre aquell píxel i el píxel diferent de zero més proper a BW (segons una determinada mètrica de distància). La funció més habitual per implementar la transformada de distància a MATLAB és *bwdist(BW,method)*. Per defecte, si no especifiquem un paràmetre com a *method*, MATLAB utilitza una mètrica de distància euclidiana: *'euclidean'*, que defineix de la forma habitual. És a dir, la distància entre (x_1, y_1) i (x_2, y_2) utilitzant *'euclidean'* és, simplement: $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. El problema és que no podem fer servir la transformada de distància euclidiana, doncs aquesta no es correspon a un recorregut des del centre d'un píxel a un altre. Per tant, haurem de fer servir una de les aproximacions de transformades que es poden especificar amb *method*: *'chessboard'*, *'cityblock'* o *'quasi-euclidean'*. De les tres mètriques possibles, la que hem d'escollir és la *'quasi-euclidean'*, ja que, d'una banda, els camins amb *'chessboard'* contenen segments horitzontals, verticals i diagonals, però la distància entre un píxel i cadascun dels seus 8 veïns és 1, mentre que els camins amb *'cityblock'* utilitzen una mesura de distància Manhattan, és a dir, es considera que un píxel només té dos veïns horitzontals i dos verticals (a distància 1, és clar). Amb la mètrica *'quasi-euclidean'*, en canvi, els

camins des d'un píxel a un altre poden consistir en segments horitzontals, verticals o diagonals, però la distància des d'un píxel fins a un veí de la seva diagonal és $\sqrt{2}$. La distància *'quasi-euclidiana'* entre (x_1, y_1) i (x_2, y_2) és: $|x_1 - x_2| + (\sqrt{2} - 1)|y_1 - y_2|$, si $|x_1 - x_2| > |y_1 - y_2|$ o $(\sqrt{2} - 1)|x_1 - x_2| + |y_1 - y_2|$, altrament.

A continuació mostrarem un petit exemple que ens permetrà il·lustrar la idea d'aplicar aquesta transformada a partir d'una imatge binària BW molt petita, com la que descriu aquesta matriu.

```
BW = 5×10
     1     1     0     0     0     0     0     0     0     0
     1     1     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     1     1
     0     0     0     0     0     0     0     0     1     1
```

Volem trobar un camí des de la regió definida pels uns de l'esquerra (marcada en verd), fins al quadrat de la dreta (en blau). Notem que la distància d'un camí entre ambdós blocs que passa per un punt intermedi qualsevol (com el que està marcat en groc) és simplement la suma de la transformada de distància del bloc verd amb la transformada de distància del bloc blau. És a dir, si definim dues matrius, $BW\_esquerra$ i BW_dreta, com:

```
BW_esquerra = 5×10
     1     1     0     0     0     0     0     0     0     0
     1     1     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
```

```
BW_dreta = 5×10
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     0     0
     0     0     0     0     0     0     0     0     1     1
     0     0     0     0     0     0     0     0     1     1
```

podem construir la matriu D, que s'obté en sumar les transformades de les dues imatges:

```
D_esquerra = bwdist(BW_esquerra, 'quasi-euclidean');
D_dreta = bwdist(BW_dreta, 'quasi-euclidean');
D = D_esquerra + D_dreta
```

```
D = 5×10 single matrix
    9.2426    8.2426    8.2426    8.2426    8.2426    8.2426    8.8284    9.4142   10.0000   11.0000
    8.8284    7.8284    7.8284    7.8284    7.8284    7.8284    7.8284    8.4142    9.0000   10.0000
    9.4142    8.4142    7.8284    7.8284    7.8284    7.8284    7.8284    7.8284    8.4142    9.4142
   10.0000    9.0000    8.4142    7.8284    7.8284    7.8284    7.8284    7.8284    7.8284    8.8284
   11.0000   10.0000    9.4142    8.8284    8.2426    8.2426    8.2426    8.2426    8.2426    9.2426
```

Notem que podem trobar un camí mínim —dels múltiples que hi ha— entre els dos objectes seguint l'element mínim de la matriu D, que és 7.8284, entre les files 2 i 4 i entre les columnes 2 i 9 de la matriu D. Podem fer ús de la funció *imregionalmin*, de MATLAB, per identificar tots aquests punts mínims de D, de manera que obtenim una nova matriu, $paths$, com aquesta:

```
paths = imregionalmin(D)
```

```
paths = 5×10 logical array
     0     0     0     0     0     0     0     0     0     0
     0     1     1     1     1     1     1     0     0     0
     0     0     1     1     1     1     1     1     0     0
     0     0     0     1     1     1     1     1     1     0
     0     0     0     0     0     0     0     0     0     0
```

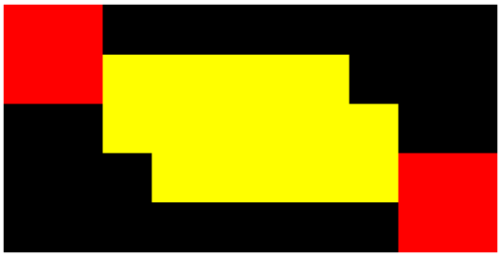

Figura 2: Conjunt de píxels que formen part dels camins mínims (en groc) entre els dos objectes (en vermell)

Com hem dit (i apreciem amb la *Figura 2*), el camí mínim entre els dos objectes de BW no és únic, es poden trobar diferents seguint diferents camins entre els dos objectes de la figura, a partir de la informació que ens dona la matriu $paths$. Per tant, hem de trobar una forma de quedar-nos amb un com a solució. Una manera de fer-ho és aplicar l'operació morfològica *thin*. Recordem que l'operació *thin* aplicada a un objecte sense forats elimina píxels de l'objecte (que volem aprimar) per a què l'objecte es converteixi, després d'un cert nombre d'aplicacions de l'operació, en un traç mínimament connectat. Podem aplicar aquesta operació sobre els píxels dels camins mínims $paths$ amb la funció *bwmorph*, de MATLAB, mitjançant la següent línia de codi. El camí mínim resultant es mostra a la *Figura 3*.

```
paths_thin = bwmorph(paths, 'thin','inf');
```

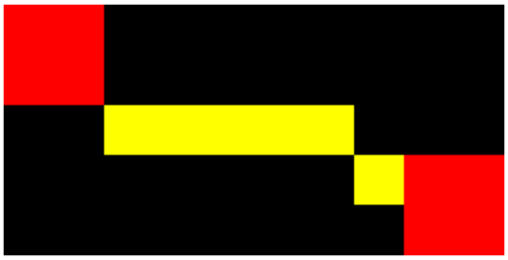

Figura 3: Un camí mínim (en groc) de longitud 6, obtingut amb l'operació 'thin'

Bé, doncs ja tenim els ingredients per enunciar un pseudo-algoritme per atacar el nostre problema inicial. La part que ocupa la cerca d'un camí mínim entre dos punts pot *resoldre's* així:

1. Calcular la transformada de distància des d'un dels punts exteriors del laberint
2. Calcular la transformada de distància des del centre del laberint (punt vermell)
3. Sumar el resultat d'ambdues transformades per crear la matriu D
4. Buscar els mínims regionals de D per construir la matriu de $paths$ i aprimar el resultat per acabar obtenint un camí mínim en concret

A continuació desenvoluparem aquest algoritme i solucionarem petits sub-problemes que presenta aquest senzill procediment. Un primer canvi, ja l'anuncio, serà l'ús de la funció *bwdistgeodesic* enlloc de *bwdist*, que hem estat fent servir, per evitar traspassar els murs del laberint.

Construcció d'un algoritme per trobar el camí mínim

1. Llegint la imatge i convertint-la a escala de grisos

Primer, llegim la imatge RGB original (*Figura 1*) i la convertim a escala de grisos.

```
I = imread('Laberint.png');
I_gray = rgb2gray(I);
```

2. Binaritzant la imatge i eliminant els punts de l'escena abans d'aplicar la transformada

Com hem vist abans, la transformada de distància calcula la distància d'un píxel amb el píxel més proper amb valor diferent de zero. Notem que la nostra intenció és calcular la transformada de distància des del centre dels cercles verd i groc de la *Figura 1*, i des del centre del cercle vermell, per tal de trobar el camí més curt que uneix aquestes dues parelles de punts, seguint l'algoritme que hem proposat al final de l'apartat anterior. Per tant, per tal de no interferir amb els càlculs de les transformades, cal que eliminem els cercles de colors de la imatge de grisos I_gray. Per

fer-ho, primer extraurem aquests cercles de la imatge original mitjançant, per exemple, una operació de *open* amb un element estructurant *disk* de radi adequat. La següent línia de MATLAB aplica aquesta operació sobre els píxels que no són del fons. El resultat es mostra a la *Figura 4*.

```
punts = imopen(I_gray < 128, strel('disk',3));
```

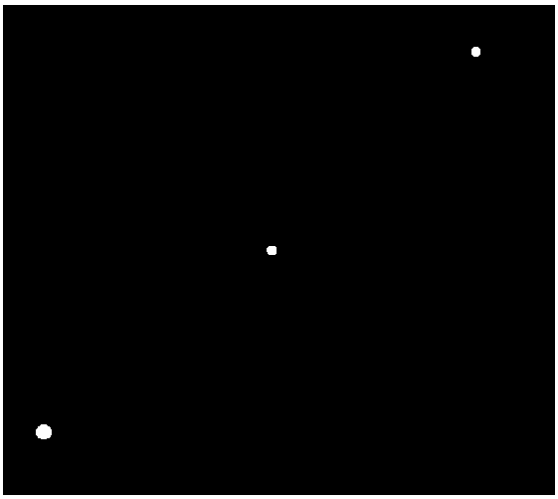

Figura 4: Punts verd, blau i vermell de l'escena original binaritzada (threshold de 128)

A continuació, binaritzarem la imatge I_gray (com ja hem fet en l'anterior crida) i li restem els punts que hem extret amb l'ultima operació (també aplicarem un *erode* per eliminar els píxels romanents dels cercles que han quedat en fer la resta), de manera que obtenim una imatge binària dels murs del laberint. Després, dilatarem aquests murs (amb un element estructurat quadrat) per evitar que el camí que trobem estigui excessivament a prop de les parets del laberint. El resultat obtingut és el que es mostra a continuació.

```
murs = (I_gray < 128) - punts;    % el·liminem els punts exteriors
murs = imerode(murs,ones(2,2));   % refinem el procés anteiror
murs = imdilate(murs, ones(5,5)); % augmentem el gruix dels murs
```

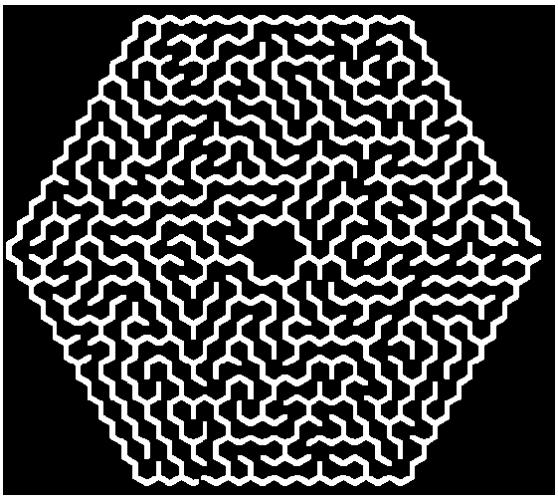

Figura 5: Murs del laberint binaritzats i dilatats després d'haver eliminat els punts de colors de la imatge original

Hem d'anar amb compte amb la mida de l'element estructurant de l'última operació de *dilate*, ja que si aquest és una mica més gran (per exemple, de 7x7), aleshores es 'tenca' l'espai entre alguns murs del laberint, entre ells, el de la porta que està al costat sud del mur més extern del laberint, produint una evident alteració del resultat.

3. Aplicant la transformada als tres punts desitjats

Primerament, hem de trobar els centres dels cercles verd, blau i vermell de la imatge original, ja que els necessitarem per calcular les transformades. Podem valdre'ns de la matriu booleana $points$ per trobar la posició dels píxels corresponents al centre d'aquests cercles, que són:

```
punt_verd = [43 441];
punt_blau = [489 49];
punt_vermell = [278 254];
```

Com ja havíem anunciat, la funció que farem servir per calcular la transformada de distància és *bwdistgeodesic*, que és similar a la funció *bwdist*, però amb diferents subtileses que fan que sigui molt adient per aquest problema. La funció *bwdistgeodesic* permet passar com a paràmetres la posició del píxel que es pren com a 'origen' (*seed point*), on la distància val zero. Així, calcularem la transformada de distància geodèsica des de cadascun dels tres centres que hem definit:

```
D_verd = bwdistgeodesic(~murs, punt_verd(1), punt_verd(2), 'quasi-euclidean');
D_blau = bwdistgeodesic(~murs, punt_blau(1), punt_blau(2), 'quasi-euclidean');
D_vermell = bwdistgeodesic(~murs, punt_vermell(1), punt_vermell(2), 'quasi-euclidean');
```

4. Calculant els camins mínims i mesurant les seves longituds

I ara, essencialment, apliquem el mateix algoritme que ja havíem vist en l'apartat anterior per trobar, d'una banda, els conjunts de píxels que contenen camins mínims entre les dues parelles de punts que estem estudiant (verd-vermell, blau-vermell), i, d'altra banda, aprimar aquests conjunts per quedar-nos amb un camí mínim:

```
D_verd_vermell = round(D_verd + D_vermell);
D_blau_vermell = round(D_blau + D_vermell);

D_verd_vermell(isnan(D_verd_vermell)) = inf;
paths_verd_vermell = imregionalmin(D_verd_vermell);
D_blau_vermell(isnan(D_blau_vermell)) = inf;
paths_blau_vermell = imregionalmin(D_blau_vermell);

path_thin_verd_vermell = bwmorph(paths_verd_vermell, 'thin', inf);
path_thin_blau_vermell = bwmorph(paths_blau_vermell, 'thin', inf);
```

Notem que ara podem mesurar la distància dels dos camins calculats a partir de les matrius *path_thin_verd_vermell* i *path_thin_blau_vermell*, ja que aquestes només contenen uns als píxels pels quals passa el camí mínim trobat entre els punts verd i vermell, i blau i vermell, respectivament (la resta de píxels són zeros). De manera que les distàncies poden trobar-se així:

```
verd_dist = sum(path_thin_verd_vermell(:) == 1)
blau_dist = sum(path_thin_blau_vermell(:) == 1)
```

5. Mostrant el camí mínim

Per últim, apliquem una operació de dilatació sobre els dos camins trobats per a poder visualitzar-los millor i utilitzem la funció *imoverlay*, que ens permet mostrar els camins (en groc) sobre la imatge original del laberint. Els resultats els exposem a la *Figura 6*.

```
P_verd = imoverlay(I, solution_verd_vermell, [1 1 0]);
imshow(P_verd)
P_blau = imoverlay(I, solution_blau_vermell, [1 1 0]);
imshow(P_blau)
```

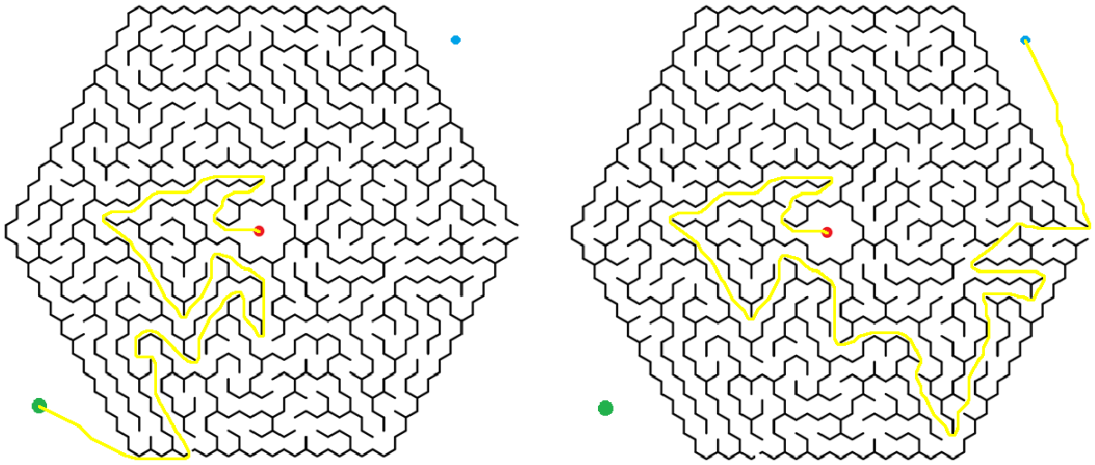

Figura 6: Camí mínim des del punt verd fins al centre (esquerra, camí de longitud 1090 píxels) i camí mínim des del punt blau fins al centre (dreta, camí de longitud 1410 píxels)

Com podem observar, els dos camins que obtenim comparteixen bona part del recorregut dins el laberint. El punt verd, però, està més a prop de l'entrada del laberint que el punt blau. La distància entre el centre del cercle verd i el centre del cercle vermell és, aproximadament, de 1090 píxels, mentre que la distància entre el punt blau i el vermell és d'uns 1410 píxels.

Per acabar, mostrem l'execució de l'algoritme amb un laberint més gran. La *Figura 7* mostra el camí (mínim) entre els punts vermells de l'escena, que no és altra més que la solució del laberint que es passa com a *input*. Notem que les entrades del laberint estan *marcades* pels cercles vermells. Cal notar que les diferents mides dels elements estructurants s'han d'ajustar d'acord a la imatge d'entrada.

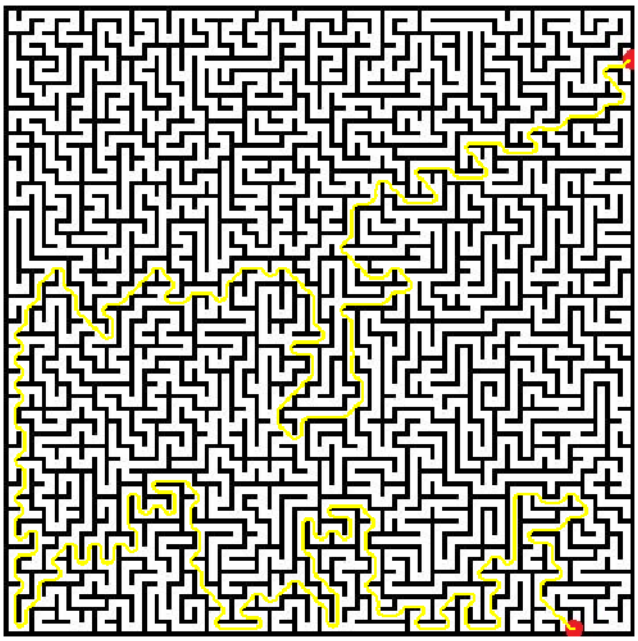

Figura 7: El camí mínim entre els dos punts vermells és la solució del laberint