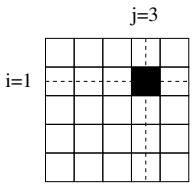
IREM de Clermont-Ferrand
Formation ISN
Algorithmique et programmation

1er mars 2012
Malika More
malika.more@u-clermont1.fr

Évaluation : Carte de distance.

Présentation.

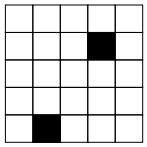
Images et pixels. On modélise une image (numérique en niveaux de gris) par une grille rectangulaire de dimensions $H \times L$. Les cases de cette grille sont appelés des pixels. Chaque pixel est repéré par ses coordonnées i et j, où i est le numéro de la ligne ($0 \leq i < H$) et j est le numéro de la colonne ($0 \leq j < L$). La valeur (couleur) de chaque pixel est un entier compris entre 0 (noir) et 255 (blanc).
Par exemple, sur l'image de taille 5×5 représentée ci-contre, la valeur du pixel de coordonnées 1 et 3 est 0, et celle de tous les autres pixels est 255.



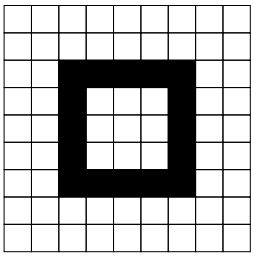
Notion de distance. Il existe plusieurs notions de « distances » entre deux pixels. On s'intéresse ici à l'une d'entre elles : la distance en taxi (ou distance de Manhattan).
La **distance en taxi** entre deux pixels est celle qu'on devrait parcourir pour aller de l'un à l'autre (sans faire de détours) dans une ville dont les rues forment un quadrillage régulier.
Plus formellement, si p et q sont deux pixels de coordonnées respectives (i_p, j_p) et (i_q, j_q), on a

$$DistTaxi(p, q) = |i_p - i_q| + |j_p - j_q|$$

Dans l'exemple ci-contre, la distance en taxi entre les deux pixels noirs est de 5.



Distance à un objet. On considère un objet constitué de certains pixels d'une grille, comme celui constitué de l'ensemble des pixels noirs ci-contre.
On dira que la distance entre un pixel p et cet objet est le minimum de la distance entre p et un pixel de l'objet. En particulier, si p est un pixel de l'objet, alors la distance entre p et l'objet est nulle.



Problème. L'objectif est de calculer la **carte de distance** d'un objet, autrement dit de déterminer la distance entre chacun des pixels p de la grille et cet objet. Le résultat obtenu dépend évidemment de la notion de distance choisie.
Par exemple, on donne ci-contre la carte de distance en taxi du carré noir.
On peut ensuite visualiser le résultat sous la forme d'une image en niveaux de gris, où la couleur d'un pixel sera d'autant plus sombre qu'il est proche de S (puisque 0 représente la couleur noire et 255 la couleur blanche).

4	3	2	2	2	2	2	3	4
3	2	1	1	1	1	1	2	3
2	1	0	0	0	0	0	1	2
2	1	0	1	1	1	0	1	2
2	1	0	1	2	1	0	1	2
2	1	0	1	1	1	0	1	2
2	1	0	0	0	0	0	1	2
3	2	1	1	1	1	1	2	3
4	3	2	2	2	2	2	3	4

Structures de données.

Image. La grille (ou l'image) G est représentée par une double liste dans laquelle la valeur de l'élément d'indice j de la liste d'indice i correspond au niveau de gris du pixel de coordonnées i et j.
Par exemple, en `Python`, la liste ci-contre correspond à l'image du carré noir sur fond blanc.
Notez que la commande `G[2][3]` va donner `0` (et le pixel en ligne 2 colonne 3 est bien noir) tandis que la commande `G[3][3]` va donner `255` (et le pixel en ligne 3 colonne 3 est blanc).

```
G=[[255,255,255,255,255,255,255,255,255],
[255,255,255,255,255,255,255,255,255],
[255,255,0,0,0,0,0,255,255],
[255,255,0,255,255,255,0,255,255],
[255,255,0,255,255,255,0,255,255],
[255,255,0,255,255,255,0,255,255],
[255,255,0,0,0,0,0,255,255],
[255,255,255,255,255,255,255,255,255],
[255,255,255,255,255,255,255,255,255]]
```

Objet. L'objet S est donné par la liste des coordonnées de ses pixels.
Par exemple, en `Python`, la liste ci-contre correspond à l'objet constitué des pixels noirs de l'image du carré noir sur fond blanc.

```
S=[[2,2],[2,3],[2,4],[2,5],[2,6],[3,2],
[3,6],[4,2],[4,6],[5,2],[5,6],[6,2],
[6,3],[6,4],[6,5],[6,6]]
```

Partie 1 : Algorithme exhaustif

1. **Algorithmique.**
   Décrire un algorithme permettant de calculer la carte de distance d'un objet S dans une grille G par une méthode exhaustive.
2. **Programmation.**
   (a) Écrire une fonction `Taxi` qui prend en entrée la liste des coordonnées des pixels de S, et les coordonnées i et j d'un pixel p et qui retourne la distances en taxi entre p et S.
   (b) En utilisant la fonction `Taxi`, écrire une fonction `CarteTaxi` qui prend en entrée la liste des coordonnées des pixels de S, et les dimensions H et L de la grille G et qui retourne la carte de distance en taxi de S.

Vous pouvez visualiser les images de test proposées et les cartes de distance obtenues à l'aide de la fonction `Affiche` donnée dans le fichier `distance.py`.

Un exemple de solution :

Pour chaque pixel (i, j) de l'image, on calcule successivement la distance en taxi entre ce pixel (i, j) et chaque pixel de S. On garde le minimum de ces valeurs, et c'est la valeur du pixel (i, j) de la $CarteTaxi$.

Un autre exemple de solution :

$CarteTaxi = [\,]$
pour i *allant de* 0 *à* $H-1$ **faire**
  $ligne = [\,]$
  pour j *allant de* 0 *à* $L-1$ **faire**
    $liste = [\,]$
    pour *chaque pixel* (a, b) *dans la liste* S **faire**
      $dist = |i-a| + |j-b|$
      $liste.append(dist)$
    fin
    $res = min(liste)$
    Ajouter à la liste $ligne$ la valeur res
  fin
  Ajouter à $CarteTaxi$ la liste $ligne$
fin

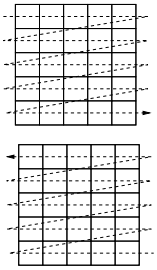
Une version très détaillée :

```
CarteTaxi = [ ] (ce sera une liste de listes comme G)
pour i allant de 0 à H − 1 (càd pour chaque ligne de l'image) faire
    ligne = [ ] (ce sera la ième ligne de CarteTaxi)
    pour j allant de 0 à L − 1 (càd pour chaque pixel de la ligne i) faire
        (pour la programmation : le bloc ci-dessous représente en fait la
        fonction Taxi)
        (on parcourt tous les pixels de l'objet S et on détermine au fur et
        à mesure la plus petite distance entre le pixel (i, j) et un pixel de
        S)
        min = H × L (on part d'une valeur qui dépasse la plus grande valeur
        possible)
        pour chaque pixel (a, b) dans la liste S faire
            dist = |i − a| + |j − b| (calcul de la distance entre les pixels (a, b) et
            (i, j))
            si dist < min (càd le pixel (a, b) est plus près du pixel (i, j) que tous
            les pixels de S traités précédemment) alors
                min = dist (càd dist est la nouvelle valeur du minimum)
            fin
            (on passe au pixel suivant de S s'il en reste)
        fin
        (on a calculé le minimum min de la distance entre le pixel (i, j) et un
        pixel de l'objet S)
        Ajouter à la liste ligne la valeur min (c'est le jème élément de la ième ligne)
        (on passe au pixel suivant de la ligne i s'il en reste)
    fin
    (on a traité tous les pixels de la ligne i)
    Ajouter à CarteTaxi la liste ligne (c'est la ième ligne du résultat)
    (on passe à la ligne suivante s'il en reste)
fin
(on a traité toutes les lignes de l'image)
```

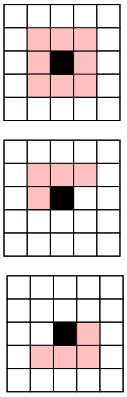
Algorithme de Ronse

Idée générale. Lorsque l'objet S est de grande taille (c'est-à-dire composé de beaucoup de pixels), l'utilisation de la fonction `Taxi` pour le calcul de la carte de distance en taxi de S allonge considérablement le temps de calcul. On présente ici un autre algorithme de calcul de cette carte, qui n'utilise pas la fonction `Taxi` : il effectue deux balayages successifs de la grille, l'un dans l'ordre lexicographique et l'autre dans l'ordre inverse, et pour chaque pixel p effectue chaque fois un calcul local qui porte uniquement sur les pixels voisins de p et qui ne dépend pas de la taille de l'objet S.
(*Référence* : `http://dpt-info.u-strasbg.fr/ cronse/TIDOC/GD/tfd.html`)
La correction de l'algorithme de Ronse (autrement dit le fait qu'il permet bien de calculer la carte de distance de l'objet S) ne va pas de soi, et ne fait pas l'objet du présent problème.
Avant de détailler l'algorithme, on a besoin de quelques notions supplémentaires.

Ordres de parcours de la grille. On parcourt la grille **dans l'ordre lexicographique** lorsqu'on commence en haut à gauche, on va de gauche à droite puis de haut en bas, et on termine en bas à droite. Autrement dit, (i, j) prend successivement les valeurs $(0, 0), (0, 1), (0, 2), \ldots, (0, L-1)$, puis $(1, 0), (1, 1), \ldots, (1, L-1)$, etc., et pour finir $(H-1, 0), \ldots, (H-1, L-1)$, comme indiqué sur le schéma ci-contre.
On parcourt la grille **dans l'ordre anti-lexicographique** lorsqu'on commence en bas à droite, on va de droite à gauche puis de bas en haut, et on termine en haut à gauche, comme indiqué sur le schéma ci-contre.



Voisinage et demi-voisinages d'un pixel Pour tout pixel p, on note $V(p)$ le **voisinage** de p, c'est-à-dire l'ensemble constitué des 8 pixels les plus proches de p et de p lui-même.
De même, on note $V^-(p)$ le **demi-voisinage antérieur** de p (autrement dit les pixels 5 du voisinage de p visités avant lui dans l'ordre lexicographique) et on note $V^+(p)$ le **demi-voisinage postérieur** de p (autrement dit les 5 pixels du voisinage de p visités après lui dans l'ordre lexicographique).
Le pixel p fait partie de son voisinage, et de ses demi-voisinages antérieurs et postérieurs.
Sur les schémas ci-dessous, on a représenté de haut en bas $V(p)$, $V^-(p)$ et $V^+(p)$ lorsque p est le pixel noir.



Masque et demi-masques On définit la distance entre un pixel et ses voisins les plus proches à l'aide d'une petite grille de taille 3×3 appelée un **masque local**. Par exemple, on montre ci-contre le masque local pour la distance en taxi.

2	1	2
1	0	1
2	1	2

La distance entre deux pixels quelconques correspond à la somme des distances données par le déplacement du masque sur un chemin de la grille de longueur minimale reliant les deux pixels.
On obtient le **demi-masque antérieur** en restreignant le masque local aux pixels rencontrés avant le pixel central dans un balayage dans l'ordre lexicographique. Pour chaque pixel q appartenant au voisinage antérieur $V^-(p)$ centré en p, on note $M_p^-(q)$ la valeur correspondante du masque local.

2	1	2
1	0	

On obtient le **demi-masque postérieur** en restreignant le masque local aux pixels rencontrés après le pixel central dans un balayage dans l'ordre lexicographique. Pour chaque pixel q appartenant au voisinage postérieur $V^+(p)$ centré en p, on note $M_p^+(q)$ la valeur correspondante du masque local.

	0	1
2	1	2

Remarque : sur les bords de la grille Lorsque le pixel central est sur un bord de la grille, certains des pixels du masque débordent de la grille. Dans ce cas, on tronque le masque en enlevant tout ce qui déborde de la grille.

Calcul de la carte de distance de S en trois étapes

- **Initialisation.** On crée une grille G de taille $H \times L$ en donnant à chaque pixel la valeur 0 s'il appartient à S et $+\infty$ sinon (concrètement, dans nos exemples on prendra $+\infty = 1000$).
- **Premier passage.** On parcourt les pixels de la grille dans l'ordre lexicographique. La nouvelle valeur donnée à chaque pixel p est obtenue par $Min\{G[i_q][j_q] + M_p^-(q) \mid q \in V^-(p)\}$.
- **Second passage.** On parcourt les pixels de la grille dans l'ordre anti-lexicographique. La nouvelle valeur donnée à chaque pixel p est obtenue par $Min\{G[i_q][j_q] + M_p^+(q) \mid q \in V^+(p)\}$.
- La grille obtenue après le second passage est la **carte de distance de S**.

Partie 2 : Algorithme de Ronse

1. **Algorithmique.**
   Écrire un algorithme réalisant chacune des trois étapes de l'algorithme de Ronse.
2. **Programmation.**
   (a) Écrire les fonctions `DemiMasqueAnterieur` et `DemiMasquePosterieur` qui prennent en entrée une grille et les coordonnées d'un pixel et qui calculent respectivement la nouvelle valeur du pixel après application du demi-masque antérieur et postérieur pour la distance en taxi. (Attention aux bords !)
   (b) Écrire la fonction `Initialisation` qui prend en entrée la liste des coordonnées des pixels de S et les dimensions H et L de la grille et qui renvoie la grille initiale.
   (c) Écrire la fonction `PremierPassage` qui prend en entrée une grille et qui effectue les calculs correspondant au premier passage à l'aide de la fonction `DemiMasqueAnterieur`.
   (d) Écrire la fonction `SecondPassage` qui prend en entrée une grille et qui effectue les calculs correspondant au second passage à l'aide de la fonction `DemiMasquePosterieur`.
   (e) Écrire la fonction `CarteRonse` qui prend en entrée la liste des coordonnées des pixels de S, et les dimensions H et L de la grille et qui retourne la carte de distance en taxi de S en utilisant l'algorithme de Ronse.

```
(Initialisation)
CarteRonse = [ ]
pour i allant de 0 à H − 1 faire
    ligne = [ ]
    pour j allant de 0 à L − 1 faire
        si le pixel (i, j) est un pixel de S alors
            val = 0
        sinon
            val = H × L
        fin
        Ajouter à la liste ligne la valeur val
    fin
    Ajouter à CarteRonse la liste ligne
fin

(Premier passage dans l'ordre lexicographique)
pour i allant de 0 à H − 1 faire
    pour j allant de 0 à L − 1 faire
        Calculer la nouvelle valeur val du pixel (i, j) après application du demi-masque antérieur à partir de la
        CarteRonse (voir détails ci-dessous)
        Donner à CarteRonse[i][j] la valeur val
    fin
fin

(Second passage dans l'ordre anti-lexicographique)
pour i allant de H − 1 à 0 par pas de −1 faire
    pour j allant de L − 1 à 0 par pas de −1 faire
        Calculer la nouvelle valeur val du pixel (i, j) après application du demi-masque postérieur à partir de
        la CarteRonse (voir détails ci-dessous)
        Donner à CarteRonse[i][j] la valeur val
    fin
fin
```

Calculer la nouvelle valeur val du pixel (i, j) après application du demi-masque antérieur à partir de la $CarteRonse$:

$B = [CarteRonse[i][j] + 0]$
si $i > 0$ **alors**
  Ajouter à la liste B la valeur $CarteRonse[i-1][j] + 1$
  si $j > 0$ **alors**
    Ajouter à la liste B la valeur $CarteRonse[i-1][j-1] + 2$
  fin
  si $j < L - 1$ **alors**
    Ajouter à la liste B la valeur $CarteRonse[i-1][j+1] + 2$
  fin
fin
si $j > 0$ **alors**
  Ajouter à la liste B la valeur $CarteRonse[i][j-1] + 1$
fin
$val = min(B)$

Calculer la nouvelle valeur val du pixel (i, j) après application du demi-masque postérieur à partir de la $CarteRonse$:

$B = [CarteRonse[i][j] + 0]$
si $i < H - 1$ **alors**
  Ajouter à la liste B la valeur $CarteRonse[i+1][j] + 1$
  si $j < L - 1$ **alors**
    Ajouter à la liste B la valeur $CarteRonse[i+1][j+1] + 2$
  fin
  si $j > 0$ **alors**
    Ajouter à la liste B la valeur $CarteRonse[i+1][j-1] + 2$
  fin
fin
si $j < L - 1$ **alors**
  Ajouter à la liste B la valeur $CarteRonse[i][j+1] + 1$
fin
$val = min(B)$