PCSI3 **DS Informatique**

17 mai 2023

2 heures 15

Les calculatrices sont interdites.

Le sujet est composé d'un exercice et d'un problème en trois parties.

L'épreuve est à traiter en langage **Python**.

Les différents algorithmes doivent être rendus dans leur forme définitive sur le Document Réponse dans l'espace réservé à cet effet en respectant les éléments de syntaxe du langage (les brouillons ne sont pas acceptés).

La réponse ne doit pas se limiter à la rédaction de l'algorithme sans explication, les programmes doivent être expliqués et commentés.

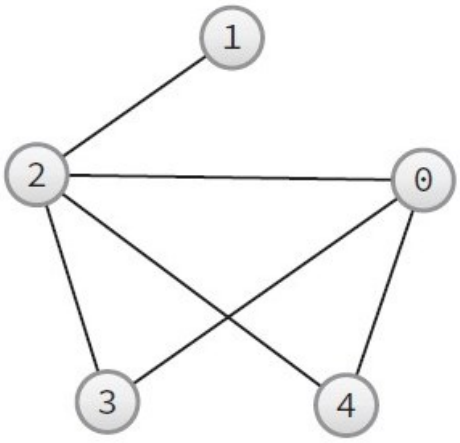
Énoncé et Annexe : 10 pages

Document Réponse (DR) : 9 pages

Seul le Document Réponse doit être rendu dans son intégralité.

Exercice : sur le chemin, être ou ne pas être...

On étudie dans cet exercice un graphe non-orienté et l'objectif est de valider si un chemin défini par une liste de sommets consécutif est envisageable dans le graphe. On considère par exemple le graphe ci-contre.



Q1. Écrire la matrice d'adjacence du graphe ci-dessus (on demande juste de représenter la matrice en expliquant sommairement : aucun code python est demandé).

Q2. Écrire en en python une fonction `testchemin(M:list, C:list)->bool` qui prend en entrée la matrice d'ajacence d'un graphe (une liste de listes) et un chemin (une liste de sommets) et qui vérifie si ce chemin est possible dans le graphe. Par exemple, sur le graphe ci-dessus, avec le chemin `[2,1,0,4]` la fonction `testchemin` doit renvoyer `False` alors qu'avec le chemin `[1,2,3]` elle doit renvoyer `True`. Cette fonction doit pouvoir s'adapter à tout graphe non-orienté défini par une matrice d'adjacence.

Problème : reconnaissance optique de caractères

Introduction

La reconnaissance optique de caractères (OCR) existe depuis de nombreuses années mais les récents travaux d'intelligence artificielle (apprentissage profond) ont considérablement augmenté les performances de la reconnaissance de documents.

L'objectif du travail proposé est de découvrir différentes étapes de la numérisation d'un document en explorant plusieurs algorithmes utilisés pour obtenir au final un document éditable conforme à l'original.

Le sujet abordera les points suivants :

- acquisition d'un document et pré-traitement dans le but d'obtenir une image numérique pertinente ;
- reconnaissance du contenu qui correspond à l'extraction du texte et de sa structure ;
- reconnaissance des caractères par identification à l'aide d'une base de données.

Partie I - Acquisition d'un document

L'acquisition du document est obtenue généralement par balayage optique. Le résultat est rangé dans un fichier de points (pixels) dont la taille dépend de la résolution.

Une image en couleurs est stockée dans une matrice `imgC` de p lignes (pixels en hauteur), q colonnes (pixels en largeur) dont chaque élément est un triplet. Chaque valeur du triplet de couleur (rouge, vert, bleu) est un entier compris entre 0 et 255. La résolution est exprimée en nombre de pixels par pouce (ppp). La valeur d'un pouce est environ égale à 2,5 cm.

Q1. Chaque entier représentant une couleur est représenté, en binaire, sous la forme d'un mot constitué de bits 0 et de 1.

Donner la taille de ce mot pour qu'il puisse représenter tous les entiers compris entre 0 et 255. Indiquer les dimensions (en pixels) d'une image en couleurs au format A4 (21 cm x 29,7 cm) pour une résolution de 300 ppp. En déduire alors la taille en octets du fichier image correspondant.

Pour diminuer la taille du document afin de pouvoir plus facilement le traiter, on réalise tout d'abord une conversion en niveaux de gris de l'image.

L'image en niveau de gris est une matrice `imgG` à p lignes et q colonnes où chaque valeur est un entier entre 0 (pixel noir) et 255 (pixel blanc).

Une matrice est représentée par une liste de listes dont les éléments de la liste interne pourront être des triplets pour les images en couleurs ou des entiers pour les images en niveau de gris.

Q2. Ecrire une fonction `dimension(img:list)->tuple` qui renvoie le triplet $(p,q,3)$ pour une image en couleurs et le triplet $(p,q,1)$ pour une image en niveau de gris. On pourra déterminer si une variable a est un entier avec la fonction `isinstance(a, int)` qui retourne True si a est un entier, `False` sinon.

Q3. Ecrire une fonction `initialise(p:int, q:int, valeur:int)->list` qui renvoie une image de dimensions (p,q) où tous les pixels sont initialisés à une même valeur `valeur`, un entier.

La formule utilisée pour déterminer la valeur d'un pixel gris en fonction des trois couleurs d'un pixel (R rouge, G vert, B bleu) est la suivante : *pixGris = 0,299∗R+0,587∗G+0,114∗B*.
On donne la fonction permettant de convertir en niveau de gris l'image en couleurs.

```
def conversion_gris(imgC:list )->list:
    p, q, nbc = dimension(imgC) #
    img = initialise(p, q, 0)
    for i in range(p):
        for j in range(q):
            r, g, b = imgC[i][j]
            val =  0.299 * r + 0.587 * g + 0.114 * b
            img[i][j] = int(val)
    return img
```

Q4. Donner la complexité de la fonction `conversion_gris(imgC:list)->list`.

La première étape du prétraitement est la **binarisation**. Cela consiste à remplacer les pixels en niveaux de gris par des pixels noirs (valeur 0) ou blanc (valeur 255) uniquement. Pour cela, la valeur du pixel gris est comparée à une valeur seuil notée `seuil`.

Q5. Proposer une fonction `binarisation(imgG:list, seuil:int)->list` qui convertit une image en niveau de gris en image en noir et blanc en imposant une valeur 255 pour tout pixel de valeur strictement supérieure au seuil, 0 sinon.

La difficulté de cette technique de binarisation est le choix de la valeur seuil pour des images ayant des problèmes d'éclairage. Une technique de restauration étudiée, dans le sujet original, peut être utilisée pour remplacer la binarisation par seuil standard.

Partie II - Reconnaissance du document

II.1 - Rotation de l'image

L'image scannée peut avoir un problème de rotation qu'il convient de corriger afin d'appliquer l'algorithme de reconnaissance des caractères (**figure 1**).

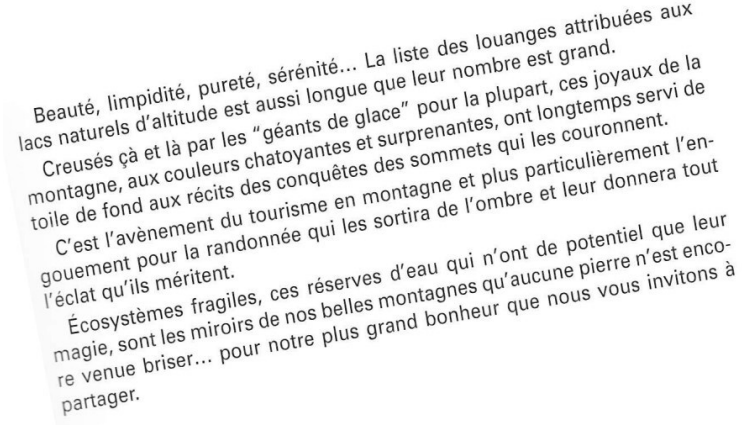

Beauté, limpidité, pureté, sérénité... La liste des louanges attribuées aux lacs naturels d'altitude est aussi longue que leur nombre est grand.
Creusés çà et là par les "géants de glace" pour la plupart, ces joyaux de la montagne, aux couleurs chatoyantes et surprenantes, ont longtemps servi de toile de fond aux récits des conquêtes des sommets qui les couronnent.
C'est l'avènement du tourisme en montagne et plus particulièrement l'engouement pour la randonnée qui les sortira de l'ombre et leur donnera tout l'éclat qu'ils méritent.
Écosystèmes fragiles, ces réserves d'eau qui n'ont de potentiel que leur magie, sont les miroirs de nos belles montagnes qu'aucune pierre n'est encore venue briser... pour notre plus grand bonheur que nous vous invitons à partager.

Figure 1 - Image avec un problème de rotation lors de l'acquisition numérique

Le paramétrage de l'image pour la rotation est donné sur la figure 2. L'image est de dimension (p,q) (possède *p* lignes et *q* colonnes). Pour faire tourner le point de coordonnées (i,j) autour du point *O*, centre de l'image, d'un angle α, on applique une rotation à l'aide d'une matrice de rotation :

$$\begin{pmatrix} n_i - p/2 \\ n_j - q/2 \end{pmatrix} = \begin{pmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{pmatrix} \begin{pmatrix} i - p/2 \\ j - q/2 \end{pmatrix}$$

On obtient alors un nouveau point de coordonnées (n_i, n_j).

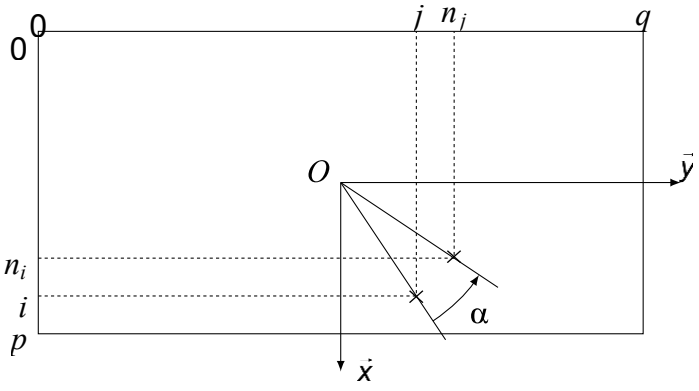


Figure 2 - Paramétrage de l'image pour la rotation

Naïvement, on pourrait penser que pour réaliser la rotation, il suffit de parcourir chaque pixel de l'image initiale en lui appliquant la rotation définie précédemment. Mais les indices étant des entiers, on se rend compte que certains pixels de la nouvelle image ne sont jamais calculés (**figure 3**) et qu'il peut apparaître des problèmes de dépassement de taille d'image.

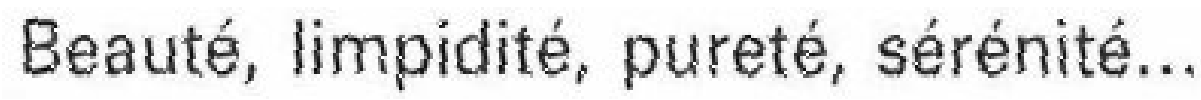

Beauté, limpidité, pureté, sérénité...

Figure 3 - Rotation naïve de l'image initiale

L'algorithme de rotation consiste donc, pour chaque pixel de la nouvelle image de coordonnées (n_i, n_j), à trouver ses coordonnées (i, j) par une rotation d'angle $-\alpha$ dans l'image initiale. La position du pixel virtuel ainsi trouvée est en fait un couple de réels (x, y). Le pixel virtuel est ainsi entouré de 4 pixels dans l'image initiale dont les abscisses sont comprises entre `int(x)` et `int(x)+1` et les ordonnées entre `int(y)` et `int(y)+1` (**figure 4**).

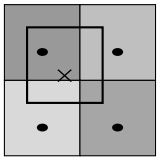


Figure 4 - Illustration du pixel trouvé entouré des 4 pixels voisins dans l'image intiale

Pour trouver la valeur du pixel virtuel, on utilise la valeur des 4 pixels voisins en réalisant une approximation bilinéaire qui consiste :

- en prenant les deux pixels voisins de la première ligne, à trouver la valeur du niveau de gris du pixel virtuel en supposant une évolution linéaire selon la coordonnée *y* entre le pixel de gauche et le pixel de droite ;
- à faire de même en prenant les pixels de la deuxième ligne ;
- enfin en travaillant sur la coordonnée *x*, à supposer une évolution linéaire entre les deux valeurs trouvées aux deux étapes précédentes.

On dispose d'une fonction :

```
lineaire(x:float, x0:int, x1:int, pix0:int, pix1:int)->float
```

qui renvoie le flottant `val`, approximation linéaire au point *x* des valeurs *pix0* prise au point x_0 et *pix1* prise au point x_1. Exemple : `lineaire(5.2, 5, 6, 100, 200)` retourne le flottant `120.`.

Q6. Choisir la fonction `bilineaire(im:list, x:float, y:float)->int` parmi les quatre propositions, données dans le **DR**, permettant de respecter les spécifications.

Le module math est importé ainsi : `import math as m`

On suppose définie une fonction : `prod_matrice_vecteur(M:list, v:list)->list`

qui renvoie le produit de la matrice `M` par le vecteur `v` sous la forme d'une liste.

Si les coordonnées du point virtuel (x, y) se situent en dehors de l'image, alors la valeur du pixel sur l'image tournée sera prise de couleur blanche, c'est-à-dire égale à 255.

Q7. Compléter la fonction `rotation(im:list, angle:float)->list` donnée dans le **DR** qui prend en argument une image en niveau de gris et un angle en degré et qui renvoie une nouvelle image tournée de l'angle `angle` donné en degré. On veillera à initialiser l'image par une image complètement blanche (pixels de valeur 255).

Une manière d'implémenter la fonction `lineaire` est la suivante :

```
def lineaire(x:float, x0:int, x1:int, pix0:int, pix1:int)->float:
    return (x-x0) * (pix1-pix0) / (x1-x0) + pix0
```

Il se trouve qu'en pratique, si on utilise des tableaux Numpy pour représenter les matrices, on peut être tenté de forcer les entiers à être du type `uint8` qui correspond à un entier non signé (d'où le « u » pour « unsigned ») stocké sur 8 bits.

Q8. Donner une raison pour laquelle il serait intéressant de se contraindre à 8 bits et expliquer le gain qu'il pourrait en découler en pratique.

Expliquer quel problème pourrait apparaître en réfléchissant au résultat de la soustraction `18-23` où 18 et 23 sont tous deux des entiers non signés sur 8 bits et où le résultat est lui aussi obligatoirement un entier non signé sur 8 bits.

En utilisant une telle structure (où `pix0` et `pix1` sont des entiers de type `uint8`), on se retrouve avec l'image pixellisée de la **figure 5**, ce qui n'est effectivement pas un résultat voulu, l'image attendue étant donnée sur la **figure 6**. L'angle choisi pour cette rotation n'est pas la valeur optimale assurant l'horizontalité du texte.

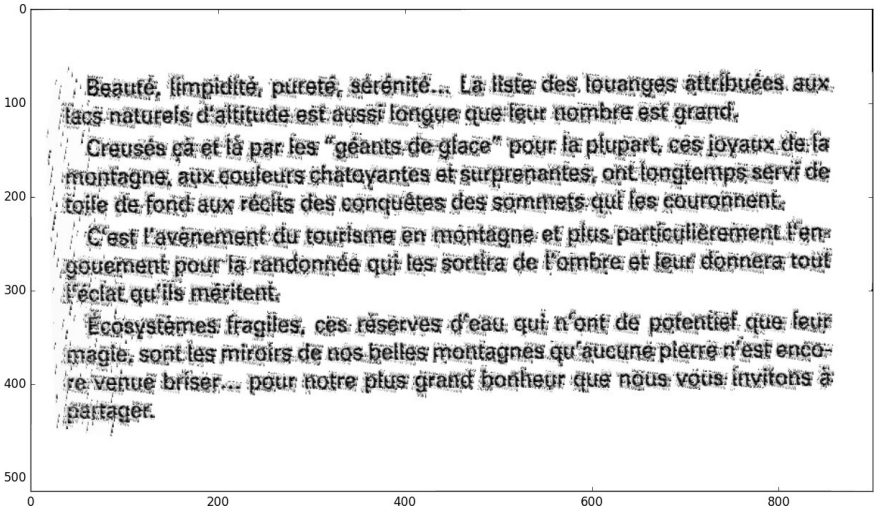


Figure 5 - Problème de pixellisation lors de la rotation

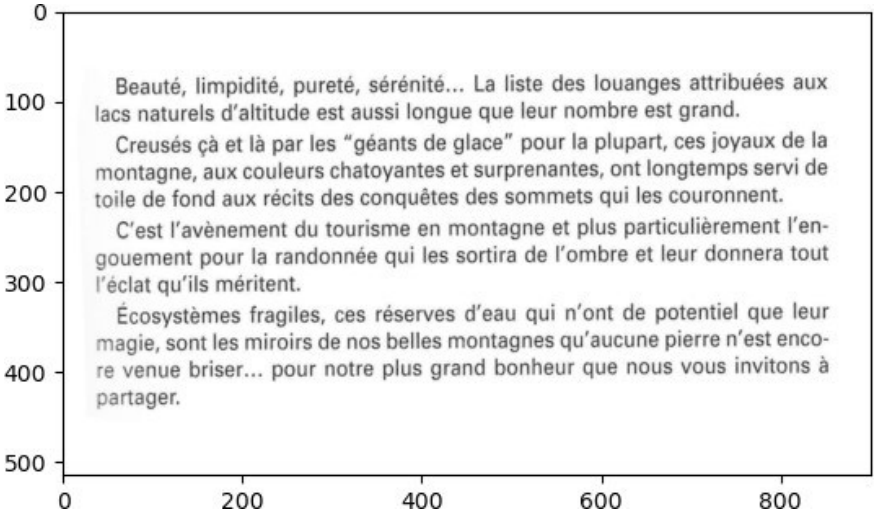


Figure 6 - Figure attendue après la rotation

II.2 - Segmentation

La segmentation consiste à découper l'image en plusieurs éléments de manière à pouvoir ensuite traiter chacun des éléments. Il faut dans l'image pouvoir dissocier les lignes, les mots puis les lettres. L'idée est de construire la liste du nombre de pixels noirs par ligne. On peut ensuite détecter les lignes en sélectionnant les zones où il y a majoritairement des pixels blancs, ce qui correspond aux zones sans texte.

On applique ensuite le même principe pour détecter les mots et les lettres en comptant les pixels blancs verticalement.

On travaille sur une image binarisée, c'est-à-dire ne contenant que des pixels blancs (255) ou des pixels noirs (0).

Q9. Proposer une fonction `histo_lignes(im:list)->list` qui prend en argument une image binarisée et renvoie une liste contenant le nombre de pixels noirs de chaque ligne sans utiliser la fonction `count`.

La fonction appliquée au texte précédent, après rotation, renvoie la liste présentée sous forme d'un histogramme sur la **figure 7**.

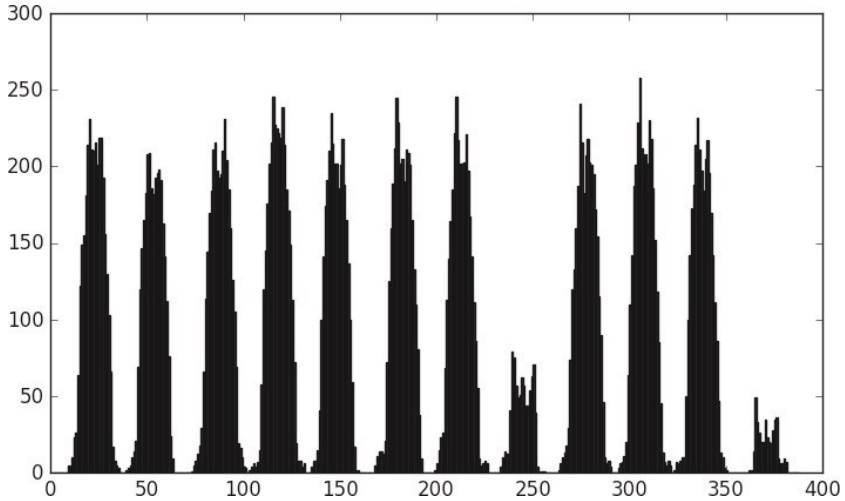


Figure 7 - Histogramme de détection des lignes

On peut observer des blocs de pixels noirs correspondant bien aux lignes. Il suffit maintenant de détecter le début d'un bloc comme étant un élément nul suivi d'un élément non nul et de détecter la fin d'un bloc comme étant un élément nul précédé d'un élément non nul.

Q10. Compléter sur le **DR** la fonction `detecter_lignes(liste:list)->list` prenant en argument une liste contenant le nombre de pixels noirs par ligne de l'image et qui renvoie une liste de couples (début ligne, fin ligne).

Cette fonction appliquée à notre exemple renvoie : `[[8, 36], [38, 64], [73, 102], [102, 132], [134, 160], [167, 193], [198, 227], [232, 257], [262, 291], [293, 322], [322, 351], [361, 382]]`.

En appliquant cette détection de ligne directement sur l'image mal orientée, il en résulte une erreur de détection. En effet, si on observe l'histogramme dans ce cas (**figure 8**), on constate qu'il n'y a plus de zones avec des pixels blancs détectées.

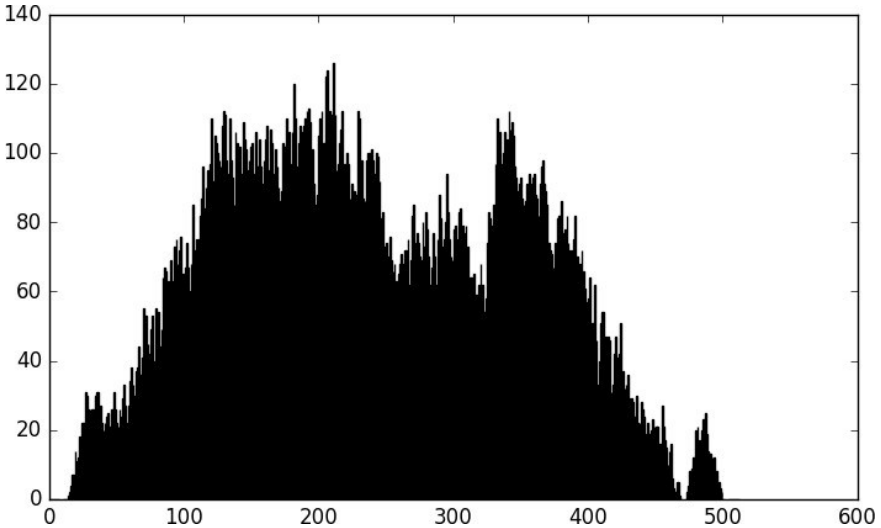


Figure 8 - Histogramme de détection des lignes sur la figure mal orientée

Il est donc nécessaire d'implanter un algorithme permettant de détecter automatiquement la bonne orientation en travaillant sur la maximisation du nombre de 0 dans la liste fournie par la fonction `histo_ligne`. On suppose que dans l'intervalle des angles de recherche, la fonction possède un unique maximum (pas d'extremum local).

L'algorithme peut être décrit de la manière suivante :

- partant d'un intervalle de départ $[a,b]$ avec les angles a et b, on calcule :
  - le nombre de 0 de la liste fournie par la fonction `histo_ligne` pour les deux orientations a et b ;
  - le nombre de 0 de la liste fournie par la fonction `histo_ligne` pour l'orientation du milieu, noté $c = (a+b)/2$;
- on itère tant que l'intervalle de recherche $[a,b]$ est plus grand qu'un epsilon donné :
  - on calcule le nombre de 0 pour l'orientation au milieu, noté ac, de l'intervalle $[a,c]$,
  - on calcule le nombre de 0 pour l'orientation au milieu, noté cb, de l'intervalle $[c,b]$,
  - on cherche où se situe le maximum entre ac, c ou cb,
  - on en déduit le nouvel intervalle de recherche, comme étant celui entourant le maximum. Par exemple, si le maximum est en c, alors le nouvel intervalle sera $[ac,cb]$.

Q11. Justifier que cet algorithme se termine.

Donner le nom de la méthode utilisée pour réaliser cet algorithme et préciser en justifiant le nombre d'itérations nécessaires pour obtenir la solution avec une précision notée ε.

Q12. Compléter la fonction `rotation_auto(im:list, a:float, b:float)->list` qui prend en argument une image `im` et les angles initiaux `a` et `b` et qui renvoie l'image avec la bonne orientation.

Après avoir séparé les lignes, en appliquant une méthode similaire, on peut extraire les caractères sur chacune de ces lignes. La **figure 9** montre les premiers caractères détectés par l'algorithme.

Figure 9 - Premiers caractères détectés par l'algorithme

Partie III - Détermination des caractères

Une fois les images de lettres isolées et débruitée (partie non traitée dans ce sujet), il s'agit de reconnaître la lettre correspondante. Différentes méthodes peuvent être employées. Nous allons étudier une méthode d'apprentissage automatique basée sur les K plus proches voisins.

Le principe de cette méthode consiste à comparer chaque caractère à un ensemble de caractères définis dans une base de données.

On dispose d'une liste `fichiers_car_ref` contenant les noms des fichiers images d'un grand nombre de caractères ayant des fontes proches de celles du texte scanné. Les fontes usuelles sont Arial, Times new roman, Calibri, Courier…

Le nom de chaque fichier, une chaîne de caractères, est défini de la manière suivante :

```
nomFonte + "_" + nomCatégorie + taillePolice + "_" + idSymbole + ".png"
```

`nomCatégorie` est un élément de la liste suivante :

```
categories = ["majuscules","minuscules","chiffres","special"].
```

`idSymbole` est un numéro qui correspond à l'indice du caractère dans un des éléments (un par catégorie) de la liste suivante :

```
symboles = ["ABCDEFGHIJKLMNOPQRSTUVWXYZ",
            "abcdefghijklmnopqrstuvwxyz",
            "0123456789",
            ".:,;'(!?)éèàçùêûâ"].
```

Exemple : `"Zurich Light BT_majuscules18_10.png"` contient l'image de la majuscule K de la police Zurich Light BT en taille 18.

On introduit la fonction suivante :

```
def lire_symbole_fichier(nomFichier:str)->str:
    car = nomFichier.split('_')
    num = car[2].split('.')[0]
    var = car[1][0:len(car[1])-2]
    ind = categories.index(var)
    return symboles[ind][int(num)]
```

Pour une liste `L`, `L.index(val)` renvoie l'indice de `val` dans la liste `L`.

Pour une chaîne de caractères `mot`, `mot.split(c)` découpe la chaîne en utilisant le caractère `c` comme séparateur et retourne la liste des sous-chaînes :

`"123,456,7".split(',')` retourne `['123', '456', '7']`

Q13. Indiquer ce que valent les variables `car`, `num`, `var`, `ind` et ce qui est renvoyé par la fonction si `nomFichier="Zurich Light BT_majuscules18_10.png"`.

Toutes les images des caractères de référence sont lues et stockées sous forme de tableaux (des listes de listes). On définit un **dictionnaire** `carac_ref` dont les clés seront les symboles apparaissant dans la liste `symboles` (par exemple "A", "a", ...). À chaque clé sera associée une liste de tableaux représentant des images de ce symbole.

La commande `img=imread(nomFichier)` permet de lire le fichier image `nomFichier` et de stocker le tableau à deux dimensions qui représente l'image dans la variable `img`.

Q14. Écrire une fonction `lire_donnees_ref(fichiers_car_ref:list)->dict` qui prend en argument la liste des noms de fichiers images `fichiers_car_ref` et qui renvoie le dictionnaire contenant tous les tableaux catégorisés.

Un des caractères à identifier est également stocké sous forme d'un tableau nommé `carac_test`. On suppose que les dimensions de ce tableau et de tous les tableaux du dictionnaire `carac_ref` sont les mêmes et qu'ils sont en niveau de gris : la couleur de chaque pixel est définie par un entier compris entre 0 (noir) et 255 (blanc).

La méthode d'identification utilisée est celle des K plus proches voisins. Dans un premier temps, elle consiste à calculer une distance entre l'image du caractère à identifier et toutes les images de référence.

En notant (i, j) les coordonnées d'un pixel dans le tableau représentant l'image, p_{ij} le pixel associé à l'image du caractère à identifier et q_{ij} celui d'un caractère de référence, on calcule pour chaque caractère de référence la distance $d = \sqrt{\sum_{i,j}(p_{ij} - q_{ij})^2}$.

Les distances d sont stockées dans un dictionnaire `distances` où, pour chaque clé égale à un symbole de la liste `symboles`, on associe une liste de distances pour chaque image de référence de ce symbole.

Q15. Écrire une fonction `distance(im1:list, im2:list)->float` qui calcule la distance entre les deux images `im1` et `im2` supposées de même dimension.

Q16. Écrire une fonction `calcul_distances(carac_ref:dict, carac_test:list)->dict` qui prend en argument le dictionnaire des tableaux catégorisés et un tableau associé au caractère à tester et qui renvoie le dictionnaire des distances.

La suite consiste à déterminer les K plus petites distances et à extraire les clés correspondantes, puis parmi ces clés, à déterminer la clé majoritaire. Une méthode envisageable est de trier les distances par ordre croissant pour prendre les K premiers éléments. On suppose qu'il y a au total n images de caractères de référence sur l'ensemble des symboles.

Q17. En se plaçant dans le pire des cas, indiquer le nom d'une méthode de tri performante envisageable, en précisant sa complexité temporelle en fonction de n.

La méthode envisagée pour extraire directement les K plus petits éléments utilise une fonction d'insertion proche de celle utilisée pour un tri par insertion.

La fonction `insere(d:float, lettre:str, voisins:list)->list` prend comme paramètre une liste `voisins` d'exactement K couples (distance, symbole), triés par ordre croissant de distance. La fonction insère le couple `(d, lettre)` dans la liste `voisins` à la bonne place, en conservant l'ordre de tri. L'insertion est faite en place.

Q18. Compléter les 5 zones manquantes des fonctions `insere` et `Kvoisins`.

Q19. Préciser la complexité temporelle asymptotique dans le pire des cas de cet algorithme en fonction de n, nombre d'images de référence, et de K. Comparer avec l'utilisation d'un tri classique sachant que n est grand et que K ne dépassera pas 5.

Q20. Écrire une fonction `symbole_majoritaire(voisins:list)->str` qui, à partir de la liste `voisins` renvoyée par la fonction `Kvoisins`, renvoie le symbole majoritaire.