CPGE PSI* 2024/2025 **Informatique**
Lycée La Fayette **Nathalie Planche**

Concours Blanc- Epreuve d'informatique

Vendredi 14 mars 2025 - Durée : 3h

N.B. : le candidat attachera la plus grande importance à la clarté, à la précision et à la concision de la rédaction. Si un candidat est amené à repérer ce qui peut lui sembler être une erreur d'énoncé, il le signalera sur sa copie et devra poursuivre sa composition en expliquant les raisons des initiatives qu'il a été amené à prendre.

RAPPEL DES CONSIGNES

- *Utiliser uniquement un stylo noir ou bleu foncé non effaçable pour la rédaction de votre composition ; d'autres couleurs, excepté le vert, peuvent être utilisées, mais exclusivement pour les schémas et la mise en évidence des résultats.*
- *Ne pas utiliser de correcteur.*
- *Écrire le mot FIN à la fin de votre composition.*

La calculatrice n'est pas autorisée.

L'épreuve est à traiter en langage Python sauf pour les bases de données.
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 : 14 pages
Document Réponse (DR) : 11 pages

Seul le Document Réponse doit être rendu dans son intégralité.

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 $\times 29,7$ cm) pour une résolution de 300 ppp . En déduire alors un ordre de grandeur de 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).
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 : $pixGris = 0,299 * R + 0,587 * G + 0,114 * B$.

De manière générale, on nomme le type $array$ pour représenter une matrice sous la forme d'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.
On introduit les fonctions :

- $dimension(img : array) -> tuple$ 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 ;
- $initialise(p : int, q : int, valeur : int) -> array$ qui renvoie une image de dimensions (p,q) où tous les pixels sont initialisés à une même valeur $valeur$.

On donne la fonction permettant de convertir en niveau de gris l'image en couleurs.

```
def conversion_gris(imgC:array)-> array :
    p,q,_ = 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
```

Q2. Donner la complexité de la fonction $conversion\_gris(imgC : array) -> array$.

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 $seuil$.

Q3. Proposer une fonction $binarisation(imgG : array, seuil : int) -> array$ 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.

La difficulté de cette technique de binarisation est le choix de la valeur seuil pour des images ayant des problèmes d'éclairage. Nous verrons que la technique de restauration étudiée par la suite 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).

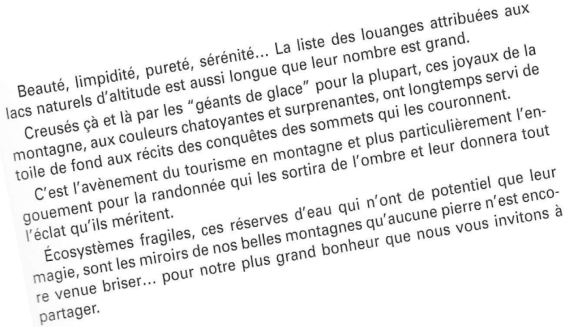

Beauté, limpidité, pureté, sérénité... La liste des louanges attribuées aux lacs naturels d'altitude est aussi longue que leur nombre est grand.
Creusés çà et là par les "géants de glace" pour la plupart, ces joyaux de la montagne, aux couleurs chatoyantes et surprenantes, ont longtemps servi de toile 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).

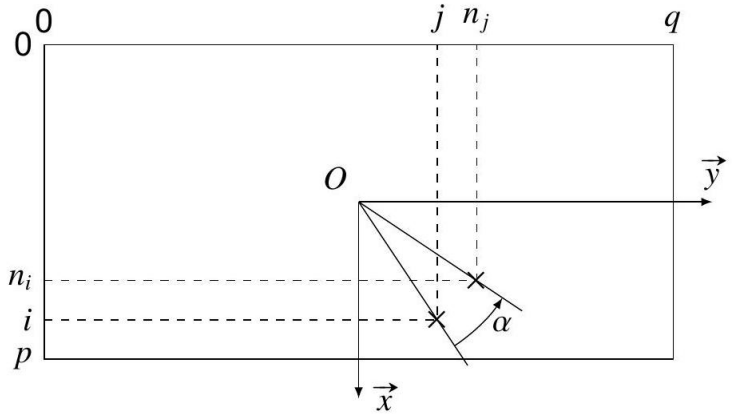


FIGURE 2 – Paramétrage de l'image pour la rotation

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.

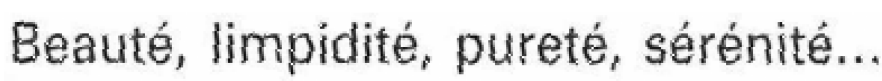

Beauté, limpidité, pureté, sérénité...

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).

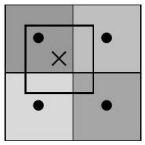


FIGURE 4 – Illustration du pixel trouvé entouré des 4 pixels voisins dans l'image initiale

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 $x0$ et $pix1$ prise au point $x1$.
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.

Q4. Choisir la fonction $bilineaire(im : array, x : float, y : float) -> array$ parmi les quatre propositions, données dans le **DR**, permettant de respecter les spécifications

Q5. Compléter la fonction $rotation(im : array, angle : float) -> array$ 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).
On suppose définie une fonction :
$prod\_matrice\_vecteur(M : array, v : list) -> list$
qui renvoie le vecteur colonne (sous forme de liste) résultat de la multiplication de la matrice M par le vecteur colonne v.

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.

Q6. 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.

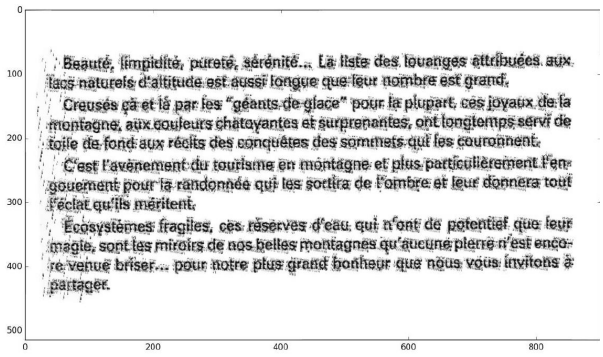


FIGURE 5 – Problème de pixellisation lors de la rotation

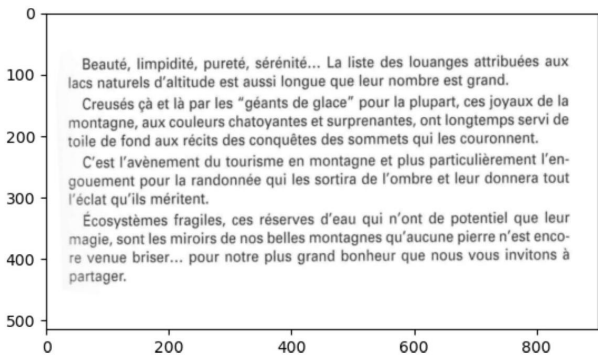


FIGURE 6 – Figure attendue après la rotation

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).

Q7. Proposer une fonction $histo\_lignes(im : array) -> list$ 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.

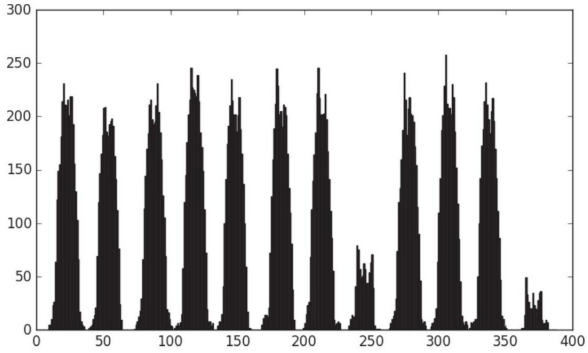


FIGURE 7 – Histogramme de détection des lignes

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.

Q8. 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.

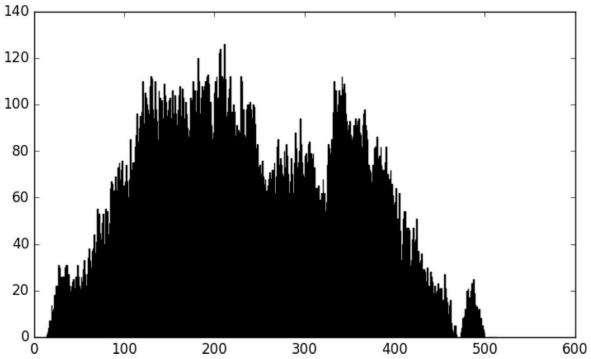


FIGURE 8 – Histogramme de détection des lignes sur la figure mal orientée

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 $c = \frac{a+b}{2}$;
- 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 $[ac,cb]$.

Q9. 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.

Q10. Compléter la fonction $rotation\_auto(im : array, a : float, b : float) -> array$ 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.

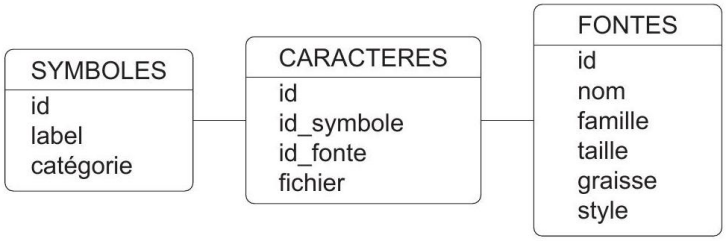
FIGURE 9 – Premiers caractères détectés par l'algorithme

Partie III - Détermination des caractères

Une fois les images de lettres isolées, 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.

III. 1 - Analyse de la base de données de caractères

La base de données contient des informations sur chaque caractère selon le type de fonte, la taille, la graisse... Trois tables sont utilisées :



La table SYMBOLES contient les attributs suivants :

- id : identifiant d'un symbole (entier), clé primaire ;
- label : nom du symbole ("A", "a", "1", "é", "!" ...) (chaîne de caractères) ;
- catégorie : parmi majuscule, minuscule, chiffre, spécial (dont accent) (chaîne de caractères).

La table CARACTERES contient les attributs suivants :

- id : identifiant d'un caractère (entier), clé primaire ;
- id_symbole : identifiant du nom du symbole (entier) ;
- id_fonte : identifiant du type de fonte (entier) ;
- fichier : nom du fichier image correspondant (chaîne de caractères).

La table FONTES contient les attributs suivants :

- id : identifiant d'une fonte (entier), clé primaire ;
- nom : nom de la fonte ("Arial", "Times new roman", "Calibri", "Zurich", ...) (chaîne de caractères) ;
- famille : nom de la famille dont fait partie la fonte ("humane", "garalde", "réale", "didone", "scripte", ...) (chaîne de caractères) ;
- taille : dimension en hauteur des caractères en pixels (entier) ;
- graisse : type de graisse ("léger", "normal", "gras", "noir", ...) (chaîne de caractères) ;
- style : type de style ("romain", "italique", "ombré", "décoratif", ...) (chaîne de caractères).

Q11. Écrire une requête SQL permettant d'extraire les identifiants des fontes dont le nom est "Zurich", de style "romain" et dont la taille est comprise entre 10 et 16 pixels.

Q12. Écrire une requête SQL permettant d'extraire tous les noms de fichiers des caractères qui correspondent au symbole de label "A".

Q13. Écrire une requête SQL permettant d'indiquer le nombre de caractères correspondant à la fonte "Zurich", de style "romain" et dont la taille est comprise entre 10 et 16 pixels groupés selon les labels des symboles.

III. 2 - Classification automatique des caractères

Dans la suite du sujet, on suppose qu'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. Le nom de chaque fichier est défini de la manière suivante :
$nomFonte + "\_" + nomCatégorie + taillePolice + "\_" + idSymbole + ".png"$

Les catégories sont définies par la liste :
$categories = ["majuscules","minuscules","chiffres","special"]$.

Les symboles considérés sont définis par la liste :
$symboles = ["ABCDEFGHIJKLMNOPQRSTUVWXYZ","abcdefghijklmnopqrstuvwxyz",$
"0123456789",". :,;' (!?)éèàçùêûâ"]. On compte 79 symboles différents.

Exemple : $ZurichLightBT\_majuscules18\_10.png$ pour 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][:len(car[1])-2]
    ind = categories.index(var)
    return symboles[ind][int(num)]
```

Pour une liste L, $L.index(val)$ renvoie la position de val dans la liste L.

Q14. Indiquer ce que valent les variables car, num, var, ind et ce qui est renvoyé par la fonction si $nomFichier = "ZurichLightBT\_majuscules18\_10.png"$.

Toutes les images des caractères de référence sont lues et stockées sous forme de tableaux $array$. 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 $array$ représentant des images.

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

Q15. 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 caractère à identifier est également stocké sous forme d'un tableau $array$ nommé $carac\_test$. On suppose que les dimensions de ce tableau et de tous les tableaux du dictionnaire $carac\_ref$ sont les mêmes.

La méthode d'identification utilisée est celle des K plus proches voisins. 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 $d = \sqrt{\Sigma_{i,j}\left(p_{ij} - q_{ij}\right)^2}$.

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.

Q16. Écrire une fonction $distance(im1 : array, im2 : array) -> float$ qui calcule la distance entre les deux images $im1$ et $im2$ supposées de même dimension.

Q17. Écrire une fonction $calcul\_distances(carac\_ref : dict, carac\_test : array) -> dict$ 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 extraire les clés correspondantes, puis parmi ces clés 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.

Q18. 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.

Une méthode plus efficace est envisagée pour extraire directement les K plus petits éléments. Elle consiste à construire par tri par insertion la liste de taille K. L'algorithme correspondant est donné dans le **DR**.

Q19. Compléter les 3 zones manquantes dans cet algorithme.

Q20. Préciser la complexité temporelle asymptotique dans le pire des cas de cet algorithme en fonction de n et de K. Comparer avec l'utilisation d'un tri classique sachant que n est grand et que K ne dépassera pas 5 .

Q21. Écrire une fonction $symbole\_majoritaire(voisins : list) -> str$ qui à partir de la liste $voisins$ renvoyée par la fonction $Kvoisins$ renvoie le symbole majoritaire.

On teste l'algorithme sur les caractères extraits dans la partie précédente ("Beauté, ").
On obtient les résultats suivants.

Nombre de voisins K	Type d'éléments dans la base de données	Nombre d'éléments dans la base n	Caractères obtenus
1	fonte similaire au texte analysé	79 images correspondant aux 79 symboles	"Bssi !-, "
4	fonte similaire au texte analysé	79 images correspondant aux 79 symboles	"Bssi !-, "
1	40 fontes proches de celle du texte analysé	40^*79 images correspondant aux 79 symboles	"Bsauté,"
4	40 fontes proches de celle du texte analysé	40^*79 images correspondant aux 79 symboles	"Bsauté, "
1	40 fontes pour 8 polices différentes	320^*79 images correspondant aux 79 symboles	"Beauté,"
4	40 fontes pour 8 polices différentes	320^*79 images correspondant aux 79 symboles	"Beauté,"

Q22. Commenter les résultats obtenus.

Partie IV - Amélioration de la reconnaissance du document : Restauration d'image

Les images de caractères peuvent être bruitées compte tenu d'une mauvaise résolution ou de parasites apparaissant pendant un scan. De même, la technique de binarisation proposée initialement ne donne pas toujours un résultat correct si le seuil est mal choisi.

La méthode du flot maximal (ou méthode de la coupe minimale) reposant sur la représentation par un graphe de l'image à restaurer est souvent utilisée pour pallier ces problèmes.

La librairie $maxflow$ disponible sous Python propose des fonctions déjà existantes pour traiter une image bruitée.

La fonction globale de traitement de l'image est la suivante :

```
import numpy
import maxflow
def graph_cut(img :array)->array:
    img = numpy.array(img) #Conversion en array de Numpy pour un usage
        plus facile ensuite
    g = maxflow.Graph[int]() #création du graphe
    nodeids = g.add_grid_nodes(dimension(img))
    g.add_grid_edges(nodeids, 5)
    g.add_grid_tedges(nodeids, img, 255-img)
    g.maxflow()
    sgm = g.get_grid_segments(nodeids)
    img2 = numpy.int_(numpy.logical_not(sgm))
    return img2
```

L'objet des questions de cette sous-partie est de comprendre chaque ligne de cette fonction et d'illustrer la méthode sur un exemple basique d'une image test (3×3) constituée de 9 pixels en niveau de gris (pixels compris entre 0 (noir) et 255 (blanc)).

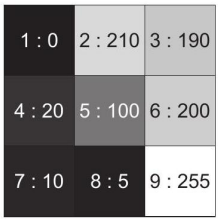


FIGURE 10 – Exemple d'image à restaurer

Les valeurs des pixels de l'exemple sont les suivantes :

```
[ [ 0, 210, 190 ],
[ 20, 100, 200 ],
[ 10, 5, 255 ] ]
```

La méthode utilise la représentation par graphe pondéré constitué de n sommets et m arêtes. Chaque sommet correspond à un pixel de l'image. $nodeids$ est donc l'ensemble des sommets du graphe correspondant à l'image de taille $dimension(img)$.

Les arêtes reliant deux sommets sont ensuite construites à l'aide de l'instruction $g.add\_grid\_edges(nodeids,5)$ entre un sommet et ses potentiels 4 voisins adjacents. À chaque arête e reliant deux sommets, un poids $w(e)$ de valeur fixe 5 est associé. Cette pondération va représenter la capacité maximale du flot définie par la suite.

Q23. Représenter le graphe correspondant à l'image de (3×3) pixels en précisant sur chaque arête la capacité maximale de 5.

Pour mettre en place la méthode de flot maximal, il est nécessaire d'introduire deux sommets supplémentaires (appelés source S et puits P) qui sont reliés par des arêtes à tous les sommets précédents. Sur chaque arête entre le sommet S et les sommets "pixels" on utilise les valeurs des pixels comme poids, et sur les arêtes entre les sommets "pixels" et le sommet P on utilise le complément à 255 des valeurs des pixels. C'est le rôle de la ligne $g.add\_grid\_tedges(nodeids, img, 255-img)$.

Q24. Compléter sur le **DR** la partie supérieure de la matrice de capacités correspondant au graphe complet de l'exemple en prenant l'ordre suivant pour les sommets : S, 1, 2,..., 9, P avec pour valeurs les poids précédemment introduits pour chaque arête. Le poids est nul si le sommet i n'est pas relié au sommet j.

La fonction $g.maxflow()$ calcule le flot maximal, ce qui permettra par la suite de partitionner les sommets.
Le flot est une notion similaire à un flux de fluide qui s'écoulerait de la source vers le puits. Mathématiquement, le flot est une fonction f définie de l'ensemble des arêtes $e \in \mathbf{E}$ vers l'ensemble des réels $\mathbb{R}$. Cette fonction vérifie les propriétés suivantes :

- $\forall e = (p,q) \in \mathbf{E}$ (avec p, q deux sommets), $f(p,q) = -f(q,p)$, le flot dans le sens q vers p est l'opposé du flot dans le sens p vers q ;
- pour tout sommet p autre que S et P : $\sum_{e=(p,.)\in\mathbf{E}} f(e) = 0$, la somme des flots arrivant et sortant d'un sommet est nulle, ce qui est similaire à la loi de Kirchoff ;
- pour toute arête $e \in \mathbf{E}, f(e) \leq w(e)$, le flot ne peut pas dépasser la capacité maximale définie initialement.

On pourrait définir une matrice de flots similaire à la matrice de capacités qui contiendrait les valeurs des flots au lieu des capacités.

On passe du graphe non orienté que nous venons de décrire à un graphe orienté. Les arêtes faisant intervenir la source sont alors orientées de la source vers les sommets (flot sortant de la source) ; celles faisant intervenir le puits sont orientées des sommets vers le puits (flot entrant dans le puits) ; les arêtes entre des sommets i et j correspondant à des pixels sont dédoublées (une de i vers j, l'autre de j vers i) et ont chacune une capacité maximale égale à 5 . La figure 11 montre un exemple de flot sur une partie seulement du graphe de l'exemple étudié. Les étiquettes de la forme i/j représentent pour i la valeur du flot et pour j la valeur de la capacité maximale.

Le flot est maximal lorsque les flots partant de la source S sont maximaux tout en respectant toutes les règles précédentes. On dit qu'une arête est saturée lorsque le flot de cette arête est égal à sa capacité.

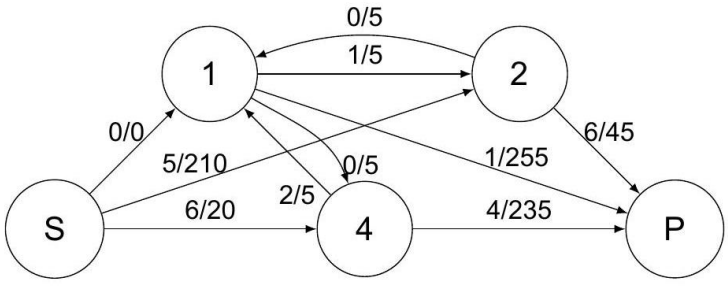


FIGURE 11 – Exemple d'extrait de graphe avec flot

Pour déterminer le flot maximal, une méthode possible consiste à saturer des arêtes. Pour cela, on utilise un graphe complémentaire appelé graphe résiduel, obtenu à partir du graphe de flot sur lequel on indique sur chaque arête $e \in \mathbf{E}$ la capacité résiduelle (dans un sens et dans l'autre) : $r(e) = w(e) - f(e)$. Si une arête est étiquetée 0 sur le graphe résiduel, alors il n'est plus possible d'emprunter cette arête pour construire le chemin de longueur minimal. La figure 12 montre le graphe résiduel associé au graphe avec flot de la figure 11.

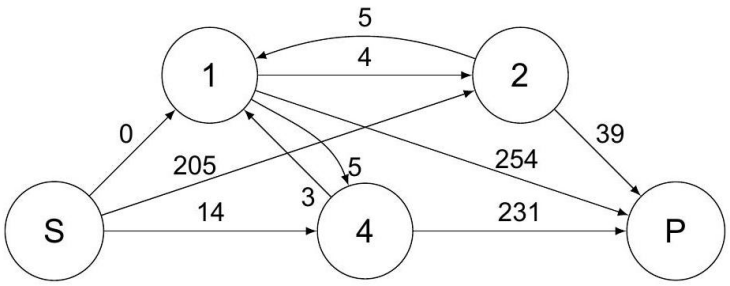


FIGURE 12 – Exemple de graphe résiduel

On utilise l'algorithme d'Edmonds-Karp : à partir du flot nul, on cherche itérativement un plus court chemin C (c'est-à-dire un chemin comportant le moins d'arêtes) de la source au puits sur lequel il n'y a pas d'arête saturée (c'est-à-dire un chemin pour lequel aucune des arêtes correspondantes du graphe résiduel n'est pondérée par 0). On rajoute alors autant de flots que possible à ce chemin (c'est-à-dire on sature l'arête qui a une capacité résiduelle minimale).

L'algorithme de recherche du flot maximal est le suivant en pseudo-code :

```
Initialisation :
poser f(e) = 0 pour toute arête e
définir le graphe résiduel initial
tant qu'il existe un chemin C de S à P dans le graphe résiduel faire
    prendre un chemin C de S à P de nombre d'arêtes minimal
    a = min(r(e) tel que e dans C)
    pour toute arête e dans C faire
        f(e)=f(e) +a
    fin pour
    mettre à jour le graphe résiduel
fin tant que
```

Q25. Appliquer cet algorithme sur les graphes du **DR** en précisant à chaque étape le chemin choisi et la valeur de l'augmentation du flot jusqu'à sa terminaison. Le graphe de gauche représente le graphe de flot et le graphe de droite le graphe résiduel. On donne le graphe initial avec un flot nul ainsi que le graphe résiduel associé.

Pour transformer l'image en niveau de gris en une image noir et blanc, c'est-à-dire pour séparer les pixels entre ceux qui prennent la valeur 0 et ceux qui prennent la valeur 255 , on va réaliser une coupe dans le graphe des pixels. On définit l'ensemble A contenant la source S et certains sommets ainsi que l'ensemble B contenant le puits P et les sommets restants.

La capacité de la coupe est la somme des capacités des arcs orientés de A vers B.
Par exemple, supposons que nous ayons coupé le graphe entre les ensembles $A = \{S, 1, 2\}$ et $B = \{P, 4\}$. En sommant les capacités maximales des arêtes orientées allant d'un sommet de A vers un sommet de B, on obtient une capacité de coupe de $20 + 5 + 255 + 45 = 325$.

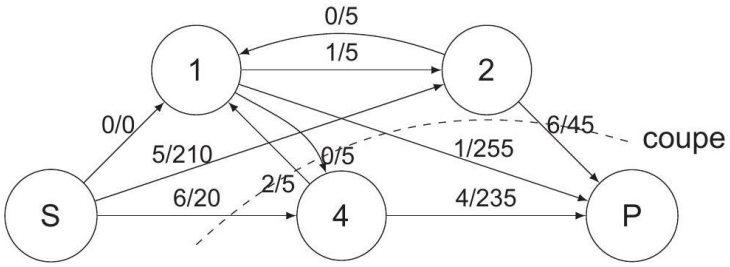


FIGURE 13 – Coupe dans un graphe

L'algorithme d'Edmonds-Karp permet de construire un flot maximum, c'est-à-dire un flot dont la somme des arêtes arrivant au puits est maximale. Le théorème du " flot maximal et coupe minimale " assure que la valeur de ce flot maximal est égale à la valeur de coupe minimale. Pour réaliser cette coupe, on met dans l'ensemble A la source S et tous les sommets accessibles, depuis S, par des arêtes non saturées ; on met dans l'ensemble B les sommets restants.

L'appel $g.get\_grid\_segments(nodeids)$ renvoie une liste indiquant, pour chacun des sommets, s'il appartient ou non au même ensemble que la source.

Q26. Dans l'exemple précédent, indiquer les deux ensembles A et B en précisant la valeur du flot maximal et en vérifiant que la capacité de coupe réalisée correspond bien à une valeur égale à celle du flot maximal.

Lorsqu'on applique la fonction précédente sur l'image d'un caractère, on obtient la nouvelle image de la figure 14.

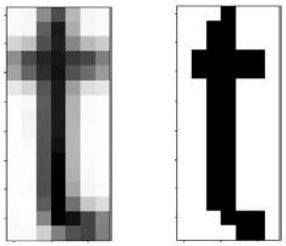


FIGURE 14 – Caractère scanné et caractère après traitement par la fonction $graph\_cut$

Q27. Indiquer pour cette image de la figure 14 à quoi correspondent les ensembles A et B. Analyser le résultat obtenu.

FIN