Concours blanc – informatique
MP
mercredi 5 novembre 2025

Lycée Camille PISSARRO

— La calculatrice est autorisée.
— Tout programme devant être complété le sera sur votre copie. Le sujet n'est pas à rendre.
— Il y a une annexe à la fin du sujet.

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 bits 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:
    n0, n1, _ = dimension(imgC)
    img = initialise(n0, n1, 0)
    for i in range(n0):
        for j in range(n1):
            r, g, b = imgC[i][j]
            val = 0.299 * r + 0.587 * g + 0.114 * b
            img[i][j] = int(val)
    return img
```

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`.

Q2. 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.

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**).

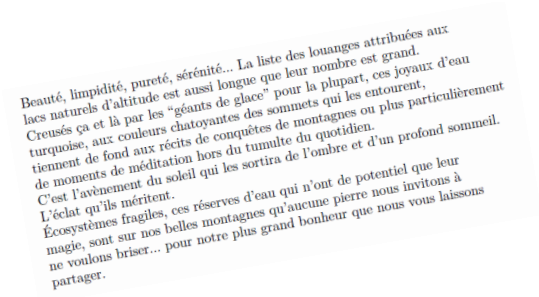
Beauté, limpidité, pureté, sérénité... La liste des louanges attribuées aux lacs naturels d'altitude est aussi longue que leur nombre est grand. Creusés ça et là par les "géants de glace" pour la plupart, ces joyaux d'eau turquoise, aux couleurs chatoyantes des sommets qui les entourent, tiennent de fond aux récits de conquêtes de montagnes ou plus particulièrement de moments de méditation hors du tumulte du quotidien. C'est l'avènement du soleil qui les sortira de l'ombre et d'un profond sommeil. L'éclat qu'ils méritent.
Écosystèmes fragiles, ces réserves d'eau qui n'ont de potentiel que leur magie, sont sur nos belles montagnes qu'aucune pierre nous invitons à ne voulons briser... pour notre plus grand bonheur que nous vous laissons 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 :

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

Ceci permet d'obtenir un nouveau point de coordonnées (n_i, n_j). Naïvement, on pourrait penser que pour

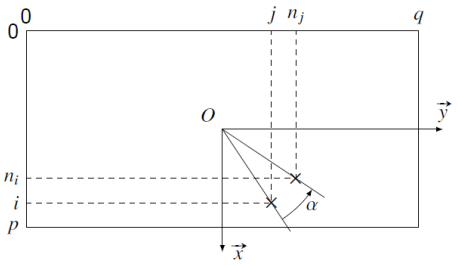

FIGURE 2 – Paramétrage de l'image pour la rotation

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 et qu'il peut apparaître des problèmes de dépassement de taille d'image.

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 3**).

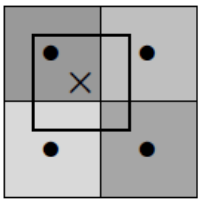

FIGURE 3 – Illustration du pixel trouvé entouré des 4 pixels voisins dans l'image initiale

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.

Q3. Choisir la fonction `bilineaire(im:array, x:float, y:float)->int` parmi ces quatre propositions, permettant de respecter les spécifications :

1. ```
   def bilineaire(im:array, x:float, y:float)->int:
       x0 = int(x)
       x1 = x0+1
       y0 = int(y)
       y1 = y0+1
       a = lineaire(y, y0, y1, im[x0][y0], im[x1][y1])
   ```
   ```

```
        b = lineaire(y, y0, y1, im[x1][y0], im[x0][y1])
        c = lineaire(x, x0, x1, a, b)
        return int(c)
```

2.
```
def bilineaire(im:array, x:float, y:float)->int:
        x0 = int(x)
        x1 = x0+1
        y0 = int(y)
        y1 = y0+1
        a = lineaire(y, y0, y1, im[x0][y0], im[x0][y1])
        b = lineaire(y, y0, y1, im[x1][y0], im[x1][y1])
        c = lineaire(x, x0, x1, a, b)
        return int(c)
```

3.
```
def bilineaire(im:array, x:float, y:float)->int:
        x0 = int(x)
        x1 = x0+1
        y0 = int(y)
        y1 = y0+1
        a = lineaire(y, y0, y1, im[x0][y0], im[x0][y1])
        b = lineaire(y, y0, y1, im[x0][y0], im[x1][y0])
        c = lineaire(x, x0, y1, a, b)
        return int(c)
```

4.
```
def bilineaire(im:array, x:float, y:float)->int:
        x0 = int(x)
        x1 = x0+1
        y0 = int(y)
        y1 = y0+1
        a = lineaire(y, y0, y1, im[x0][y0], im[x1][y1])
        b = lineaire(y, y0, y1, im[x0][y0], im[x1][y1])
        c = lineaire(y, y0, y1, a, b)
        return int(c)
```

Q4. Compléter la fonction `rotation(im:array, angle:float)->array` donnée en fin de question 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.

```
def rotation(im:array, angle:float)->array:
    p, q, _ = dimension(im)
    imr = ...................
    angr = ...................
    matR = ...................
    for ni in range(...................):
        for nj in range(...................):
            x, y = ..................................................
            x = x + ...........
            y = y + ...........
            if ..............................:
                ..................................
    return imr
```

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
```

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).
Q5. 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.
La fonction appliquée au texte précédent, après rotation, renvoie la liste présentée sous forme d'un histogramme sur la **figure 4**.

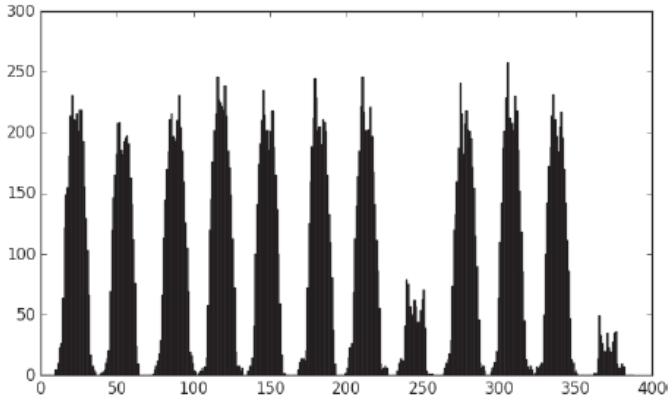

FIGURE 4 – Histogramme de détection des lignes

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.

Q6. Recopier et compléter la fonction `detecter_lignes(liste:list)->list` ci-dessous 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]]`.

```
def detecter_lignes(liste: list) -> list:
    lignes = []
    i = 0
    deb = -1 # contient -1 tant qu'on parcourt des lignes de pixel blanc
    n = len(liste)
    while .....................:
        # début d'une suite de lignes contenant des pixels noirs
        if liste[i] != 0 and deb == -1:
            deb = i
        # fin d'une suite de lignes contenant des pixels noirs
        elif liste[i] == 0 and deb != -1:
            ...................
            deb = ..............
        if i == n - 1 and liste[i] != 0:
            lignes.append([deb, n-1])
        i += 1
    return lignes
```

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 5**), on constate qu'il n'y a plus de zones avec des pixels blancs détectées.

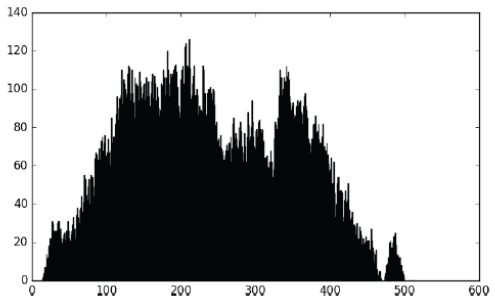

FIGURE 5 – Histogramme de détection des lignes sur la figure mal orientée

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 $[ac, cb]$.

Q7. 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.

Q8. 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.

```
def nb_zeros(im:array, angle:float)->int:
    imr = rotation(im, angle)
    ligne = histo_ligne(imr)
    f = ligne.count(0)
    return f
def rotation_auto(im:array, a:float, b:float)->array:
    c = (a+b)/2
    fc = nb_zeros(im, c)
    while b-a > 0.1: # plus grand que 0.1 degré
        ac = ............
        fac = ............
        cb = ............
        fcb = ............
        maxi = max(fac, fc, fcb)
        if ............ == maxi:
            b = c
            c = ac
```

```
            fc = fac
        elif ............ == maxi:
            a = ac
            b = cb
        else:
            a = ............
            c = ............
            fc = ............
    return rotation(im, (b+a)/2)
```

Après avoir séparé les lignes, en appliquant une méthode similaire, on peut extraire les caractères sur chacune de ces lignes.

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.

SYMBOLES	CARACTERES	FONTES
id	id	id
label	id_symbole	nom
catégorie	id_fonte	famille
	fichier	taille
		graisse
		style

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).

Q9. 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.

Q10. *(5/2 uniquement)* Écrire une requête SQL permettant d'extraire tous les noms de fichiers des caractères qui correspondent au symbole de label "A".

Q11. *(5/2 uniquement)* 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.

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", ";,:.!()?èàçùêâ".`
On compte 79 symboles différents.

Exemple : `Zurich Light BT_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.

Q12. 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 `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`.

Q13. 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

$$d = \sqrt{\sum_{i,j} (p_{ij} - q_{ij})^2}$$

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.

Q14. Écrire une fonction `distance(im1:array, im2:array)->float` qui calcule la distance entre les deux images `im1` et `im2` supposées de même dimension.

Q15. 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.

Q16. *(Question bonus)* 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 le suivant :

```
def Kvoisins(distances:dict, K:int)->list:
    voisins = [(float("inf"), "") for k in range(K)]
    for lettre in distances:
        d = distances[lettre]
        for j in range( ..... ):
            if ..... :
                k = len(voisins) - 1
                while ..... :
                    voisins[k] = voisins[k-1]
                    k = k - 1
                voisins[k] = [d[j], lettre]
    return voisins
```

Q17. *(Question bonus)* Compléter les 3 zones manquantes dans cet algorithme.

Q18. 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	"Bss1 !,-"
4	fonte similaire au texte analysé	79 images correspondant aux 79 symboles	"Bss1 !,-"
1	40 fontes proches de celle du texte analysé	40*79 images correspondant aux 79 symboles	"Beauté,"
4	40 fontes proches de celle du texte analysé	40*79 images correspondant aux 79 symboles	"Beauté,"
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,"

Q19. Commenter les résultats obtenus.

ANNEXE

Rappels des syntaxes en Python

Fonctionnalités	**Python**
détermination du nombre de zéros dans la liste X	`X.count(0)`
définir une chaîne de caractères	`mot = 'Python'`
taille d'une chaîne	`len(mot)`
extraire des caractères (avec le même fonctionnement des indices que pour les extractions de sous-listes)	`mot[2:7]`
éliminer le `\n` en fin d'une ligne	`ligne.strip()`
découper une chaîne de caractères selon un caractère passé en argument. On obtient une liste qui contient les caractères séparés. Dans l'exemple ci-contre, on découpe à chaque occurrence du caractère ","	`mot.split(',')`
ouverture d'un fichier en lecture et lecture des données (`data` est une liste de chaînes de caractères dont la taille est le nombre de lignes du fichier lu)	`with open('nom_fichier','r') as f:` `data = f.readlines()`

FIN