Épreuve d'Informatique – Session 2025 – Filière MP

Les candidats sont informés que la précision des raisonnements algorithmiques ainsi que le soin apporté à la rédaction et à la présentation des copies seront des éléments pris en compte dans la notation. Il convient en particulier de rappeler avec précision les références des questions abordées. Si, au cours de l'épreuve, un candidat repère ce qui peut lui sembler être une erreur d'énoncé, il le signale sur sa copie et poursuit sa composition en expliquant les raisons des initiatives qu'il est amené à prendre.

Remarques générales :

- ✓ Cette épreuve est composée d'un exercice et de deux parties tous indépendants ;
- ✓ Toutes les instructions et les fonctions demandées seront écrites en Python ;
- ✓ Les questions non traitées peuvent être admises pour aborder les questions ultérieures ;
- ✓ Toute fonction peut être décomposée, si nécessaire, en plusieurs fonctions.

***Important* : Le candidat doit impérativement commencer par traiter toutes les questions de l'exercice ci-dessous, et écrire les réponses dans les premières pages du cahier de réponses.**

Exercice : (4 points)

Q.1- Écrire la fonction ***fact (k)*** qui reçoit en paramètre un entier positif ***n***, et qui retourne la valeur de ***k!***.

NB : ***k! = 1*2*3* … *k*** et ***0! = 1***

Q.2- Déterminer la complexité de la fonction ***fact (k)***, avec justification.

Q.3- Écrire la fonction ***som_fact (n)*** qui reçoit en paramètre un entier positif ***n***, et qui retourne la valeur de la somme suivante : ***0! + 1! + 2! + … + n!***

Q.4- Écrire la fonction ***liste_fact (L)*** qui reçoit en paramètre une liste ***L*** de nombres entiers, et qui retourne une nouvelle liste ***F*** contenant les valeurs des factorielles des éléments de ***L***.

Exemple : ***liste_fact ([10, 5, 12, 9])*** retourne la liste ***[10!, 5!, 12!, 9!]***

Q.5- Écrire la fonction ***existe (x, L)*** qui reçoit en paramètres une liste ***L*** de nombres entiers positifs, et un entier positif ***x***. La fonction retourne ***True*** si ***x!*** existe dans ***L***, sinon la fonction retourne ***False***.

Exemple : ***existe (5, [350, 100, 120, 719])*** retourne ***True***, car ***5! = 120***

Q.6- Écrire la fonction ***compte (L, p)*** qui reçoit en paramètres une liste ***L*** de nombres entiers positifs, et un entier positif ***p***. La fonction retourne le compte des éléments de ***L*** qui sont supérieurs strictement à ***p!***.

Exemple : ***compte ([350, 100, 130, 719], 5)*** retourne ***3***, car ***350 > 5!***, ***130 > 5!*** *et* ***719 > 5!***.

Q.7- Écrire la fonction ***superieure (L, k)*** qui reçoit en paramètres une liste ***L*** de nombres entiers positifs, et un entier positif ***k***. La fonction retourne ***True*** si tous les éléments de ***L*** sont supérieurs strictement à ***k!***, sinon, la fonction retourne ***False***.

Partie I :

Apprentissage automatique supervisé

Reconnaissance optique de chiffres

L'**OCR** (**R**econnaissance **O**ptique de **C**aractères) désigne les procédés informatiques pour la traduction d'images de textes imprimés en fichiers texte. L'idée est de partir d'une image représentant un chiffre ou une lettre et d'en extraire le contenu. Pour y parvenir, l'une des méthodes est l'algorithme KNN (*K Nearest Neighbors, K plus proches voisins*).

On dispose de plusieurs images de chiffres manuscrits ***0, 1, 2, …, 9***. Chaque image correspond à un chiffre manuscrit. Plusieurs images peuvent représenter le même chiffre (voir figure ci-dessous).

A- Base de données et langage SQL

Les informations sur les images des chiffres manuscrits sont rangées dans une base de données composée d'une seule table : ***'images'***.

La table ***'images'*** est composée de deux champs :

- Le champ ***nom*** (type texte) contient le nom de l'image d'un chiffre manuscrit. ;
- Le champ ***chiffre*** (type entier) contient le chiffre représenté par l'image.

Exemples :

nom	***chiffre***
image100	**7**
image273	**4**
image13	**1**
image94	**7**
image1425	**0**
image8610	**7**
image77	**2**
…	…

Q.8- Écrire, en algèbre relationnelle, une requête qui donne pour résultat les noms des images qui correspondent au chiffre **5**.

Q.9- Écrire une requête SQL qui donne pour résultat les noms des images qui correspondent au chiffre **5**, et les noms des images qui correspondent au chiffre **8**, triés dans l'ordre alphabétique.

Q.10- Écrire une requête SQL qui donne pour résultat les noms des images, tels que le nom est composé d'au moins **6** caractères.

Q.11- Écrire une requête SQL, qui donne pour résultat le compte des images.

Q.12 - Écrire une requête SQL, qui donne pour résultat les chiffres et le compte des images correspondantes à chaque chiffre. Le résultat doit être trié dans l'ordre décroissant des comptes des images.

Q.13- Écrire une requête SQL, qui donne les chiffres tels que le compte des images correspondantes à chaque chiffre est supérieur à ***10%*** du compte de toutes les images.

B- Algorithme KNN

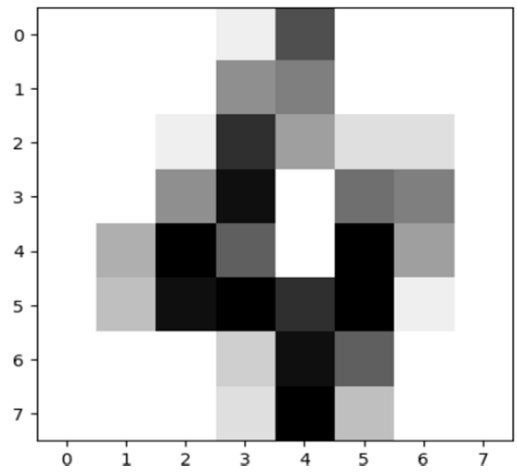
Dans cette partie, on suppose que le module **numpy** est importé :

```
import numpy as np
```

Dans la suite de cette partie, l'image de chaque chiffre manuscrit est représentée par une matrice carrée de ***8*** lignes et ***8*** colonnes.

Exemple :

L'image suivante correspond au chiffre **4**. Elle est représentée par la matrice ***M*** suivante :




```
[[ 0  0  0  2 14  2  0  0]
 [ 0  0  0 14  8  0  0  0]
 [ 0  0 10  9  0  4  4  0]
 [ 0  4 14  1  1 15  8  0]
 [ 0  4 16  5 11 16  2  0]
 [ 0  6 16 16 16 11  0  0]
 [ 0  0  4  0 12  6  0  0]
 [ 0  0  0  1 13  1  0  0]]
```

L'algorithme KNN est un algorithme de Machine Learning qui appartient à la classe des algorithmes d'apprentissage supervisé, et qui peut être utilisé pour résoudre les problèmes de classification ou de régression.

L'algorithme KNN reçoit un ensemble de données qui est étiqueté avec des valeurs de sorties correspondantes sur lequel il va pouvoir s'entraîner et définir un modèle de prédiction. Cet algorithme pourra par la suite être utilisé sur de nouvelles données afin de prédire leurs valeurs de sorties correspondantes.

Phase d'entrainement

Dans cette phase, on utilise deux variables globales ***E*** et ***F***, telles que :

- La variable globale ***E*** est initialisée par une liste qui contient plusieurs images de chiffres manuscrits : chaque élément E_i est une matrice carrée **8** lignes **8** colonnes qui représente un chiffre manuscrit.

  Exemples :

  - La matrice E_0 représente une image manuscrite du chiffre **4** ;
  - La matrice E_1 représente une image manuscrite du chiffre **3** ;
  - La matrice E_2 représente une image manuscrite du chiffre **9** ;
  - **…**

- La variable globale ***F*** est initialisée par une liste de même taille que ***E***. Chaque élément F_i est un entier compris entre ***0*** et ***9***, qui correspondant à l'image représentée par la matrice carrée E_i.

  Exemple :

  F= [4, 3, 9, 7, 1, 7, 6, 7, 6, 3, 6, 2, 1, 5, 4, 9, 4, 3, 9, 9, 2, 7, 7, 0, 4, 7, 4, 1, 2, 8, 3, 1, …]

  - La matrice E_0 représente une image manuscrite du chiffre $F_0 = 4$;
  - La matrice E_1 représente une image manuscrite du chiffre $F_1 = 3$;
  - La matrice E_2 représente une image manuscrite du chiffre $F_2 = 9$;
  - **…**

On suppose que ***X*** est une liste qui représente l'image un chiffre manuscrit quelconque. Pour prédire le chiffre correspondant à l'image ***X***, le principe de l'algorithme ***KNN*** est le suivant :

- Pour la mesure de similarité, on définit une fonction qui calcule la distance entre deux images ;
- On trie les éléments de ***E*** dans l'ordre croissant de la distance entre ***X*** et chaque élément de ***E*** ;
- On choisit un entier strictement positif ***k*** ;
- Parmi les ***k*** images de ***E*** les plus proches à ***X***, on retient le chiffre correspondant au plus grand nombre d'images.

Q.14- On considère deux matrices carrées ***A*** et ***B*** qui représentent deux images de deux chiffres manuscrits. Pour la mesure de similarité entre ***A*** et ***B***, on propose d'utiliser la distance de *Manhattan* définie par la formule suivante :

$$\boldsymbol{distance}\ (\boldsymbol{A},\boldsymbol{B}) = \sum_{\boldsymbol{i}=\boldsymbol{0}}^{\boldsymbol{7}} \sum_{\boldsymbol{j}=\boldsymbol{0}}^{\boldsymbol{7}} |\boldsymbol{A_{i,j}} - \boldsymbol{B_{i,j}}|$$

Écrire la fonction ***distance (A, B)*** qui retourne la distance de *Manhattan* entre ***A*** et ***B***.

Q.15- Écrire la fonction ***indices ( )*** qui retourne la liste ***D*** des indices des éléments de ***E***.

Exemple :

indices ( ) retourne la liste ***D =*** **[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, …]**

Q.16- Écrire la fonction ***tri_indices (D, X)*** qui reçoit en paramètres la liste ***D*** des indices des éléments de ***E***, et une matrice ***X*** qui représente l'image d'un chiffre manuscrit quelconque. La fonction trie les éléments D_i de ***D*** dans l'ordre croissant de la distance de Manhattan entre ***X*** et chaque élément dans ***E*** d'indice D_i. (Ne pas utiliser la fonction *sorted()* et la méthode *sort()*)

Exemples :

X est une matrice carrée qui représente un chiffre manuscrit.

Après l'appel de la fonction ***tri_indices (D, X)***, on obtient la liste des indices suivante :

[38, 15, 27, 24, 7, 67, 160, 51, 87, 10, 19, 8, 93, 13, 11, 44, 12, 9, …]

À partir de cette liste triée, on déduit que :

- L'image E_{38} est la première plus proche image à l'image ***X*** ;
- L'image E_{15} est la deuxième plus proche image à l'image ***X*** ;
- L'image E_{27} est la troisième plus proche image à l'image ***X*** ;
- **…**

Q.17- Écrire la fonction de ***plus_proches (X, k)*** qui reçoit en paramètres une matrice ***X*** qui représente l'image d'un chiffre manuscrit quelconque, et ***k*** un entier strictement positif. La fonction retourne la liste des ***k*** chiffres de ***F*** correspondants aux ***k*** images de ***E*** les plus proches à ***X***.

Exemple :

X est une matrice carrée qui représente un chiffre manuscrit.

plus_proches (X, 15) retourne la liste des chiffres : **[8, 8, 8, 8, 8, 8, 8, 1, 8, 3, 2, 8, 8, 8, 2]**

Q.18- Écrire la fonction de ***knn (X, k)*** qui reçoit en paramètres une matrice ***X*** qui représente l'image d'un chiffre manuscrit quelconque, et ***k*** un entier strictement positif. En utilisant le principe de l'algorithme ***KNN***, la fonction retourne le chiffre de ***F*** correspondant au plus grand nombre d'images, parmi les ***k*** images de ***E*** les plus proches à ***X***.

Exemple :

X est une matrice carrée qui représente un chiffre manuscrit.

knn (X, 15) retourne le chiffre ***8***

Phase du test

Dans cette phase, on utilise deux variables globales ***T*** et ***P***, telles que :

- La variable globale ***T*** est initialisée par une liste qui contient plusieurs images de chiffres manuscrits : chaque élément ***Ti*** est une matrice carrée **8** lignes **8** colonnes.
- La variable globale ***P*** est initialisée par une liste de même taille que ***T***. Chaque élément ***Pi*** est un entier compris entre ***0*** et ***9***, qui correspondant à l'image représentée par la matrice ***Ti***.

*Pour mesurer la précision du modèle, on calcule le pourcentage des images de **T**, telles que le chiffre représenté par chaque image est le même que le chiffre prédit par le modèle KNN.*

Q.19- Écrire la fonction de ***precision (k)*** qui reçoit en paramètre un entier ***k*** strictement positif et qui retourne la précision du modèle.

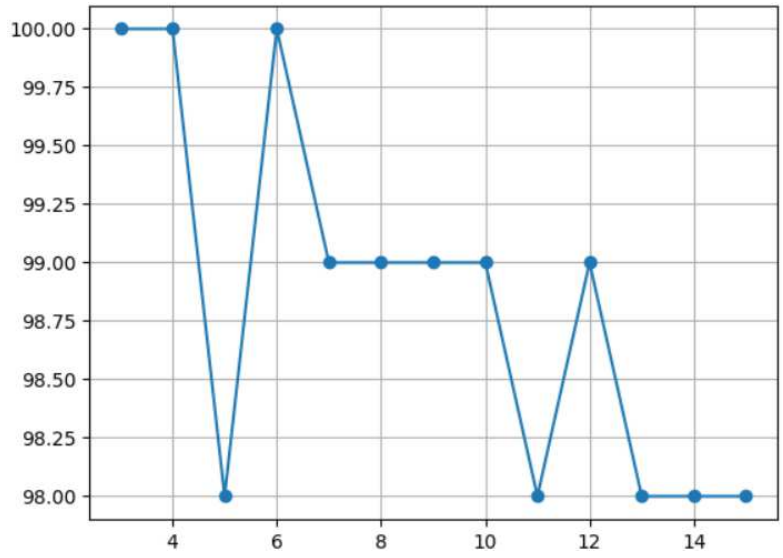
Exemples :

- ***precision(5)*** retourne **98 %**
- ***precision(10)*** retourne **99 %**

Q.20- On suppose que le module **matplotlib.pyplot** est importé :

```
import matplotlib.pyplot as plt
```

Écrire le programme python qui permet de tracer la représentation suivante, qui correspond à la précision du modèle pour chaque valeur de ***k = 3, 4, 5, …, 15***.



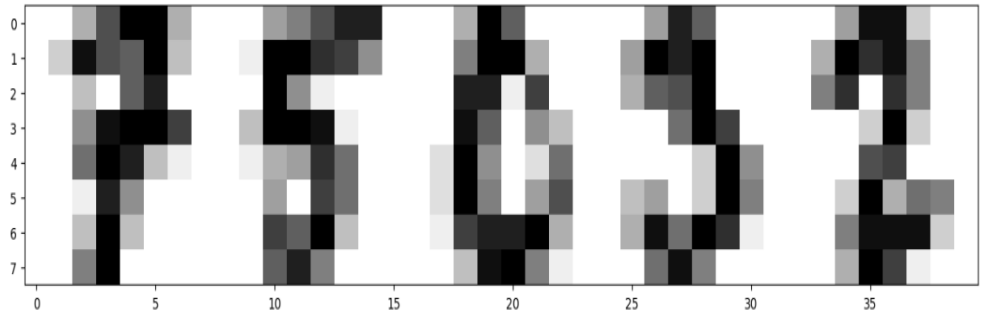
C- *Reconnaissance automatique du code postal*

La reconnaissance automatique de l'écriture est un domaine de recherche qui a trouvé une application à grande échelle dans le tri du courrier. Les enveloppes passent devant une caméra, et chaque image est traitée automatiquement par une machine qui localise le code postal et le reconnaît.

On considère une matrice ***M*** qui représente l'image d'un code postal, telle que :

- Le nombre de lignes de ***M*** est ***8*** ;
- Le nombre de colonnes de ***M*** est un multiple de ***8*** ;
- Chaque chiffre est représenté par une sous-matrice de ***M*** de ***8*** lignes et ***8*** colonnes.

Exemple :



L'image ci-dessus est représentée par la matrice de **8** lignes et **40** colonnes :

```
[[ 0   0   5  11  16  16   5   0   0   0   6   8  11  14  14   0  ... ]
 [ 0   3  15  11  10  16   4   0   0   1  16  16  13  12   7   0  ... ]
 [ 0   0   4   0  10  14   0   0   0   0  16   7   1   0   0   0  ... ]
 [ 0   0   7  15  16  16  12   0   0   4  16  16  15   1   0   0  ... ]
 [ 0   0   9  16  14   4   1   0   0   1   5   6  13   9   0   0  ... ]
 [ 0   0   1  14   7   0   0   0   0   0   6   0  12   9   0   0  ... ]
 [ 0   0   4  16   4   0   0   0   0   0  12  10  16   4   0   0  ... ]
 [ 0   0   8  16   0   0   0   0   0   0  10  14   8   0   0   0  ... ]]
```

Q.21- Écrire la fonction de ***code_postal (M, k)*** qui reçoit en paramètres la matrice ***M*** qui représente un code postal, et un entier ***k*** strictement positif. En utilisant l'algorithme ***knn***, la fonction retourne la valeur du nombre qui correspond au code postal.

Exemple :

code_postal (M, 4) retourne le nombre ***75032***

Partie II :

Réduction d'image
Algorithme 'Seam Carving'

Dans cette partie, on manipule des matrices de nombres entiers positifs.

Exemple :

```
M = [ [238,  83,  20, 120,  25, 240,  46,   5,  46, 153],
      [238, 152, 110, 251, 148,  20, 246, 144, 251, 203],
      [ 96, 197, 169,  12,  47, 166,  37, 229, 115,  57],
      [216,   9,  49,  12, 152, 206, 195, 164,  34, 108],
      [ 85,  44,   4,  94, 153, 225, 195, 198,  18, 121],
      [ 14, 159,  96, 152,  30, 204, 182, 146, 244, 238],
      [ 40,  92,  67, 167,   9, 125, 127, 154, 152,  72],
      [ 32, 137, 191, 143,  33, 113,  72, 187, 173,  36] ]
```

Pour plus de clarté, dans la suite de cette partie, on utilisera cette matrice dans les exemples des fonctions de la section (A) ci-dessous.

A- *Chemin vertical dans une matrice*

Dans une matrice ***M***, un chemin vertical est un chemin du haut vers le bas, qui part d'une case dans la première ligne de ***M*** jusqu'à une case dans la dernière ligne de ***M***.

Depuis une case $M_{i,j}$ (qui ne se trouve pas sur la dernière ligne), les seuls déplacements autorisés sont :

- Si $M_{i,j}$ est sur la première colonne, on peut se déplacer vers $M_{i+1,j}$ ou bien vers $M_{i+1,j+1}$.
- Sinon si $M_{i,j}$ est sur la dernière colonne, on peut se déplacer vers $M_{i+1,j-1}$ ou bien vers $M_{i+1,j}$.
- Sinon, on peut se déplacer vers $M_{i+1,j-1}$ *ou bien* vers $M_{i+1,j}$ ou bien vers $M_{i+1,j+1}$.

Pour représenter un chemin vertical, on utilise une liste **C** telle que :

- La taille de ***C*** est égale au nombre de lignes de ***M*** ;
- Les éléments de ***C*** sont des indices de colonnes dans ***M***.

Par exemple, on considère le chemin vertical qui part de la case $M_{0,2}$, et qui passe par les cases suivantes :

```
[ [238,  83,  20, 120,  25, 240,  46,   5,  46, 153],
  [238, 152, 110, 251, 148,  20, 246, 144, 251, 203],
  [ 96, 197, 169,  12,  47, 166,  37, 229, 115,  57],
  [216,   9,  49,  12, 152, 206, 195, 164,  34, 108],
  [ 85,  44,   4,  94, 153, 225, 195, 198,  18, 121],
  [ 14, 159,  96, 152,  30, 204, 182, 146, 244, 238],
  [ 40,  92,  67, 167,   9, 125, 127, 154, 152,  72],
  [ 32, 137, 191, 143,  33, 113,  72, 187, 173,  36] ]
```

Le chemin $M_{0,2}$ (=*20*) → $M_{1,3}$ (=*251*) → $M_{2,3}$ (=12) → $M_{3,2}$ (=*49*) → $M_{4,1}$ → $M_{5,2}$ → $M_{6,3}$ → $M_{7,4}$ (=*33*) est représenté par la liste ***C = [2, 3, 3, 2, 1, 2, 3, 4]***

La somme des valeurs des cases par les quelles passe un chemin est appelée : ***coût*** du chemin.

Par exemple, le coût du chemin ci-dessus est : ***20 + 251 + 12 + 49 + 44 + 96 + 167 + 33 = 672***

Q.22- Écrire la fonction ***cout_chemin (M, C)*** qui reçoit en paramètres une matrice ***M*** de nombres entiers, et une liste ***C*** qui représente un chemin vertical dans ***M***. La fonction le coût du chemin ***C***.

Exemple :

cout_chemin (M, [2, 3, 3, 2, 1, 2, 3, 4]) retourne ***672*** = $M_{0,2}+M_{1,3}+M_{2,3}+M_{3,2}+M_{4,1}+M_{5,2}+M_{6,3}+M_{7,4}$

Ils existent plusieurs chemins verticaux qui partent d'une cases $M_{0,j}$. Dans le but de trouver le chemin vertical de coût minimal, la méthode ***'glouton'*** consiste à se déplacer, à chaque embranchement, vers la case de plus petite valeur.

Q.23- Écrire la fonction ***glouton (M, j)*** qui reçoit en paramètres une matrice ***M*** de nombres entiers, et un entier ***j*** qui représente l'indice d'une colonne dans ***M***. En utilisant la méthode ***'glouton'***, la fonction retourne une liste qui représente le chemin vertical de plus petit coût dans ***M***, et qui part de la case $M_{0,j}$.

Exemple : ***glouton (M, 2)*** retourne le chemin ***[2, 2, 3, 3, 2, 2, 2, 1]*** *(son coût est 458)*

Q.24- Est-ce que la méthode 'glouton' permet de trouver le chemin vertical de plus petit coût ?

Justifier votre réponse.

Pour trouver le coût minimal de tous les chemins verticaux, qui partent d'une case $M_{0,j}$, on peut utiliser la méthode ***'Top-Down'*** (la mémoisation) de la programmation dynamique.

Q.25- Écrire la fonction ***cout_min (M, i, j)*** qui reçoit en paramètres une matrice ***M*** de nombres entiers, et deux entiers ***i*** et ***j*** qui représentent respectivement l'indice d'une ligne et l'indice colonne dans ***M***. En utilisant le principe de *la mémoisation*, la fonction retourne le minimum des coûts de tous les chemins verticaux, qui partent d'une case $M_{i,j}$.

Exemple :

cout_min (M, 0, 2) retourne ***320***

Dans toute la matrice ***M***, la recherche du chemin vertical de coût minimal est un problème qu'on peut résoudre en utilisant la méthode ***'Bottom-Up'*** de la programmation dynamique, dont le principe est le suivant *:*

On commence par créer une matrice ***R*** de même dimension que ***M*** : **Chaque élément $R_{i,j}$ représente le coût du chemin vertical minimal qui part de la case $M_{i,j}$**. Les éléments de ***R*** sont calculés du bas vers le haut, à l'aide de l'algorithme suivant :

- Si l'élément $R_{i,j}$ se trouve sur la dernière ligne de **R**, alors $R_{i,j} \leftarrow M_{i,j}$
- Sinon si l'élément $R_{i,j}$ se trouve sur la première colonne de **R**, alors $R_{i,j} \leftarrow M_{i,j} + \min(R_{i+1,j}, R_{i+1,j+1})$
- Si l'élément $R_{i,j}$ se trouve sur la dernière colonne de **R**, alors $R_{i,j} \leftarrow M_{i,j} + \min(R_{i+1,j-1}, R_{i+1,j})$
- Si l'élément $R_{i,j}$ ne se trouve ni sur la première colonne, ni sur la dernière colonne de **R**, alors $R_{i,j} \leftarrow M_{i,j} + \min(R_{i+1,j-1}, R_{i+1,j}, R_{i+1,j+1})$

Q.26- Écrire la fonction ***couts (M)*** qui reçoit en paramètre une matrice ***M*** de nombres entiers. En utilisant l'algorithme ci-dessus, la fonction retourne la matrice ***R***.

Exemple :

couts (M) retourne la matrice ***R*** suivante :

[[625, 383, 320, 420, 270, 485, 291, 617, 658, 799],
[473, 387, 300, 441, 338, 245, 714, 612, 694, 646],
[235, 336, 308, 190, 225, 484, 468, 615, 501, 443],
[346, 139, 179, 178, 318, 431, 492, 516, 386, 460],
[171, 130, 198, 166, 225, 297, 441, 532, 352, 467],
[86, 231, 220, 194, 72, 246, 340, 334, 352, 346],
[72, 124, 204, 200, 42, 158, 199, 226, 188, 108],
[32, 137, 191, 143, 33, 113, 72, 187, 173, 36]]

Q.27- Écrire la fonction ***chemin_min (M)*** qui reçoit en paramètre une matrice ***M*** de nombres entiers. La fonction retourne la liste qui représente le chemin vertical de coût minimal dans la matrice ***M***.

Exemple :

[[625, 383, 320, 420, **270**, 485, 291, 617, 658, 799],
[473, 387, 300, 441, 338, **245**, 714, 612, 694, 646],
[235, 336, 308, 190, **225**, 484, 468, 615, 501, 443],
[346, 139, 179, **178**, 318, 431, 492, 516, 386, 460],
[171, 130, 198, **166**, 225, 297, 441, 532, 352, 467],
[86, 231, 220, 194, **72**, 246, 340, 334, 352, 346],
[72, 124, 204, 200, **42**, 158, 199, 226, 188, 108],
[32, 137, 191, 143, **33**, 113, 72, 187, 173, 36]]

chemin_min (M) retourne le vecteur ***[4, 5, 4, 3, 3, 4, 4, 4]***.

(son coût est : $M_{0,4} + M_{1,5} + M_{2,4} + M_{3,3} + M_{4,3} + M_{5,4} + M_{6,4} + M_{7,4} = 25+20+47+12+94+30+9+33 = 270$)

Q.28- Écrire la fonction ***supprime_chemin (M, C)*** qui reçoit en paramètre une matrice ***M*** de nombres entiers, et ***C*** un vecteur qui représente un chemin vertical dans ***M***. La fonction supprime dans la matrice ***M,*** les éléments correspondants au chemin ***C***.

Exemple :

Après l'appel de la fonction ***supprime_chemin (M, [4, 5, 4, 3, 3, 4, 4, 4])***, la matrice ***M*** devient ainsi :

[[238, 83, 20, 120, 240, 46, 5, 46, 153],
[238, 152, 110, 251, 148, 246, 144, 251, 203],
[96, 197, 169, 12, 166, 37, 229, 115, 57],
[216, 9, 49, 152, 206, 195, 164, 34, 108],
[85, 44, 4, 153, 225, 195, 198, 18, 121],
[14, 159, 96, 152, 204, 182, 146, 244, 238],
[40, 92, 67, 167, 125, 127, 154, 152, 72],
[32, 137, 191, 143, 113, 72, 187, 173, 36]]

B- *Réduction d'une image par la méthode 'Seam Carving'*

Une image numérique en **niveaux de gris** est représentée par une matrice à deux dimensions. Chaque élément de la matrice est appelé **pixel**. La valeur de chaque pixel est un entier compris entre **0** et **255**, ainsi **l**'image est codée sur **256** niveaux de gris :

- Le nombre **0** correspond au noir, et le nombre **255** correspond au blanc ;
- Et chacun des autres nombres **1**, **2**, **3**,…, **253**, **254** correspond à un niveau de gris entre le noir et le blanc.

Exemple :

L'image en niveaux de gris ci-dessus, est représentée par la matrice ***M*** suivante *(533 lignes, 800 colonnes)* :

[[129, 139, 77, 100, 127, 78, 86, 105, 98, 103, 121, 106, 100, 131, 142, 61, 71, 105, 125, 140, …],
[78, 78, 98, 106, 127, 105, 44, 60, 93, 84, 103, 93, 126, 158, 174, 106, 55, 72, 123, 100, …],
[60, 66, 125, 129, 142, 136, 57, 51, 58, 74, 70, 96, 91, 150, 152, 145, 80, 91, 122, 67, …],
[84, 110, 129, 141, 141, 132, 112, 77, 111, 122, 127, 81, 66, 78, 129, 173, 182, 137, 135, 90, …],
[64, 100, 100, 124, 130, 136, 165, 129, 60, 117, 134, 145, 83, 100, 76, 121, 163, 202, 154, 93, …],
…
]

Pour plus de clarté, dans la suite de cette partie, on utilisera cette image dans les exemples des fonctions.

Le redimensionnement est une transformation applicable à une image numérique qui consiste à en modifier la taille, que ce soit pour l'agrandir ou pour la rétrécir, comme le ferait un zoom.

Le **seam carving**, ou recadrage intelligent, est un algorithme de réduction d'image développé par Shai Avidan et Ariel Shamir en 2007. Cet algorithme réduit une image par la suppression de chemins de pixels dits de moindre énergie, ce qui permet de moins la déformer.

L'énergie d'un pixel est en général mesurée par son contraste comparé à ses plus proches voisins, mais d'autres techniques, comme la détection de forme, peuvent être utilisées.

Calcul du gradient de l'image sur chaque pixel :

On définit une matrice ***G*** de même dimension que ***M***. Chaque élément $G_{i,j}$ correspond à l'intensité du gradient de l'image sur le pixel $M_{i,j}$: plus il est grand, plus il y a de forte variation de couleur au niveau de ce pixel.

Pour calculer les éléments de la matrice des gradients ***G***, on utilise la formule (à droite) correspondante à chaque élément de ***G*** (à gauche) :

G =

1	5	…	5	5	2
7	9	9	…	9	8
⋮	9	…	9	⋮	8
7	⋮	9	…	9	⋮
7	9	…	9	9	8
3	6	6	…	6	4

1. $G_{i,j} = |M_{i,j} - M_{i+1,j}| + |M_{i,j} - M_{i,j+1}|$
2. $G_{i,j} = |M_{i,j} - M_{i+1,j}| + |M_{i,j} - M_{i,j-1}|$
3. $G_{i,j} = |M_{i,j} - M_{i-1,j}| + |M_{i,j} - M_{i,j+1}|$
4. $G_{i,j} = |M_{i,j} - M_{i,j-1}| + |M_{i,j} - M_{i-1,j}|$
5. $G_{i,j} = |M_{i,j} - M_{i+1,j}| + |M_{i,j+1} - M_{i,j-1}|$
6. $G_{i,j} = |M_{i,j} - M_{i-1,j}| + |M_{i,j+1} - M_{i,j-1}|$
7. $G_{i,j} = |M_{i,j} - M_{i,j+1}| + |M_{i-1,j} - M_{i+1,j}|$
8. $G_{i,j} = |M_{i,j} - M_{i,j-1}| + |M_{i-1,j} - M_{i+1,j}|$
9. $G_{i,j} = |M_{i+1,j} - M_{i-1,j}| + |M_{i,j+1} - M_{i,j-1}|$

Le numéro de chaque élément $G_{i,j}$ *(à gauche) correspond au numéro de la formule (à droite) permettant de calculer la valeur de cet élément* $G_{i,j}$.

Q.29- Écrire la fonction ***gradients (M)*** qui reçoit en paramètre une matrice ***M*** qui représente une image en niveaux de gris. La fonction retourne la matrice des gradients ***G*** correspondante à ***M***.

Pour réduire une image en niveau de gris ***M****, la méthode* ***'Seam Carving'*** *consiste à :*

- ***a.*** *Construire la matrice des gradients* ***G*** *correspondante à* ***M*** *;*
- ***b.*** *Trouver le chemin vertical* ***C*** *de plus petit coût dans la matrice* ***G*** *;*
- ***c.*** *Supprimer le chemin* ***C*** *dans l'image* ***M****.*

Q.30- Écrire la fonction ***reduction (M, n)*** qui reçoit en paramètre une matrice ***M*** qui représente une image en niveaux de gris, et ***n*** un entier strictement positif. En utilisant l'algorithme *Seam Carving*, la fonction supprime dans l'image ***M*** les pixels de ***n*** chemins verticaux de plus petits coûts dans la matrice des gradients correspondante à ***M***.

Exemple :

L'image suivante est obtenue après la suppression de ***100*** chemins verticaux.

********** **FIN** **********