Annales
corrigées et commentées

Concours
2023 / 2024 / 2025

MP MPi
PC PSi

Informatique

CCINP & e3a
Mines-Ponts
Centrale-Supélec
X

ellipses

Maxime Berger

Chapitre 1

Quelques fonctions utiles

Les algorithmes de cette section sont très fréquents dans les sujets de concours, vous trouverez presque toujours une question qui vous demande de reconnaître, d'écrire ou de modifier un de ces algorithmes. Si vous les travaillez spécifiquement, vous gagnerez un temps précieux et une petite décharge de confiance lorsque vous tomberez sur ces questions aux écrits.

Échanger deux variables

Le premier algorithme que je vous propose est celui qui échange le contenu de deux variables `a` et `b`. Un algorithme simple, mais qui peut demander du temps si on doit y réfléchir pendant l'écrit. Bien sûr, la version naïve

```
a = b
b = a
```

ne donne pas le résultat attendu, puisque vos variables `a` et `b` contiennent maintenant la même valeur.

L'astuce consiste à poser une variable intermédiaire `c` qui gardera en mémoire la valeur de `a` avant de la modifier.

```
c = a
a = b
b = c
```

En python spécifiquement, vous avez le droit à une version abrégée :

```
a, b = b, a
```

(Attention, ceci est interdit dans les autres langages.)

L'idée a l'air simple, presque bête, pourtant beaucoup de candidats s'y perdent.

Prenez une minute pour réfléchir à la meilleure façon d'échanger le contenu de 3 variables de manière circulaire, c'est-à-dire que `a` prend la valeur de `b`, `b` prend la valeur de `c` et `c` prend la valeur de `a`.

De combien de variables temporaires avez-vous besoin ?

Une seule variable temporaire suffit.

Détaillons une application un peu plus élaborée issue du sujet Mines-Ponts 2025 :
la variable `L` contient une liste de nombres qu'on souhaite trier par ordre décroissant.
Saurez-vous reconnaître l'analogie avec ce qui précède et la variable qui joue le rôle de `c` ?

```
for i in range(1, len(L)):
    x = L[i]
    j = i
    while j > 0 and L[j-1] < x:
        L[j] = L[j-1]
        j -= 1
    L[j] = x
```

À chaque tour de la boucle `for`, on va placer l'élément `L[i]` à la bonne position parmi les éléments déjà triés `L[0]`, `L[1]`, ..., `L[i-1]`. Pour cela,

- On compare `L[i]` avec `L[i-1]` et si `L[i-1] < L[i]`,
  on échange les deux variables `L[i]` et `L[i-1]`.
- On compare ensuite `L[i]` avec `L[i-2]` et si `L[i-2] < L[i]`,
  on échange les deux variables `L[i]` et `L[i-2]`.
- etc, jusqu'à la bonne place pour `L[i]`.

À la fin des échanges, on a fait remonter d'un rang tous les `L[k]` vérifiant `L[k] < L[i]` et on a créé un trou à la position `j`. On récupère alors la valeur de `L[i]` qu'on avait stockée dans la variable `x` et on la place à cet endroit.

C'est donc la variable `x` qui joue le rôle de `c`.

Les questions concernant ce type d'algorithmes ne sont pas réservées aux premières questions faciles des sujets.
Allez lire le sujet du concours Mines-Pont 2024 filière MPI, épreuve 2 : Toute la dernière partie du sujet consiste à trouver les bons échanges à réaliser pour obtenir toutes les permutations possibles de n éléments.

Manipulation de listes

C'est un sujet fondamental : en général, les premières questions d'un sujet vous demanderont un algorithme qui parcourt une liste. Le but pourra être de faire la somme (ou la moyenne) des éléments de la liste, de trouver un élément spécifique, de calculer des moyennes glissantes, ...

Si vous n'êtes pas à l'aise avec les listes (ou les tableaux), commencez par ouvrir un fichier python, tapez

```
L = [1, 2, 3, 4, 5]
```

et écrivez :

- Une fonction pour calculer la somme des éléments de la liste,
- Une fonction pour calculer la différence entre le premier élément et le dernier,
- Une fonction pour renvoyer une nouvelle liste où chaque élément est multiplié par 2.
- Une fonction pour renvoyer une nouvelle liste où l'élément i est la somme de $L[i]$ et de $L[i+1]$, où $i+1$ est pris modulo n.

Si vous ne pouvez pas écrire ces algorithmes en 5 minutes, avancer dans les sujets sera difficile. Lisez les conseils à la fin de ce chapitre, nous parlerons d'une méthode pour progresser.
Si vous y êtes parvenus, écrivez maintenant ces fonctions avec une boucle `while`, puis de manière récursive.

Les éléments d'une liste vérifient-ils une condition ?

Dans plusieurs sujets, on vous donne une liste d'éléments, et on vous demande de vérifier

- soit qu'un élément de la liste vérifie une certaine condition.
- soit que tous les éléments de la liste vérifient une certaine condition.
- soit qu'il existe un élément de la liste qui ne vérifie pas une certaine condition.

Les algorithmes qui répondent à ces trois problématiques se ressemblent furieusement, et diffèrent seulement par certains aspects : l'idée est de poser une variable booléenne `test` initialisée avec une valeur adéquate, puis de parcourir la liste et de modifier cette variable en fonction de la condition.

Par exemple, dans l'épreuve commune de CCINP 2024, à la question 11, on vous donne un dictionnaire `jeu` qui contient une clé `plateau` associée à un tableau de 12 entiers. Écrivez un algorithme qui vérifie qu'au moins une case parmi les 6 dernières cases du plateau contient un entier strictement positif.

Allez-y, faites-le.

L'algorithme est le suivant :

```
test = False
for i in range(6, 12):
    if jeu['plateau'][i] > 0:
        test = True
return test
```

Ici, la variable `test` est initialisée à `False`, si on trouve un élément strictement positif, on change `test` à `True`.

Nous n'aurions d'ailleurs même pas besoin d'attendre la fin de la boucle, dès qu'on trouve une case vérifiant la condition, on peut arrêter de chercher et renvoyer `True` :

```
for i in range(6, 12):
    if jeu['plateau'][i] > 0:
        return True
return False
```

Si on a atteint la fin de la boucle sans qu'aucun élément du tableau ne vérifie la condition, on renvoie la valeur `False`.

Si on vous demande de vérifier que tous les éléments du tableau vérifient une condition, on initialiserait cette fois `test` à `True` :

```
test = True
for x in L:
    if not condition(x):
        test = False
return test
```

Pendant le parcours, on met `test` à `False` dès qu'on trouve un élément qui ne vérifie pas la condition. On pourrait écrire une version similaire à la précédente, et arrêter la boucle dès qu'on trouve un élément qui ne vérifie pas la condition.

La première quesion du concours Mines Ponts pour le filière MP-option info 2024 demande exactement un algorithme de ce type :

On vous donne une relation binaire sur les entiers de 0 à $n-1$.
Cette relation est représentée par un tableau à double entrée `R` de taille $n \times n$ tel que

`R[i][j]` vaut `True` si i est en relation avec j et `False` sinon.

Écrivez alors un algorithme pour vérifier que cette relation est antisymétrique.

Allez-y, voyez-vous le parallèle avec les algorithmes ci-dessus ?

Il suffit de tester que les coefficients diagonaux de la matrice sont tous égaux à `True`.

```
def check_reflexivity(r):
    n = len(r)
    reflexive = True
    for i in range(n):
        if not r[i][i]:
            reflexive = False
    return reflexive
```

Le sujet demande le programme en langage `ocaml`, mais c'est le même algorithme.

Voyons un autre classique des questions de concours :

Déterminer un maximum

La question est présente dans la quasi-totalité des sujets :
On vous donne une liste de valeurs (ou d'objets qui possèdent un attribut numérique) et on vous demande d'écrire un algorithme qui va retourner la valeur maximale (ou minimale).

La version « de base » suit les étapes suivantes :

1. On commence par initialiser une variable `maxi` avec une valeur :
   - soit très petite, souvent $-\infty$
   - soit une valeur inférieure à tous les éléments de la liste, par exemple -1 si toutes les valeurs sont positives
   - soit la première valeur de la liste.
2. On parcourt les éléments de la liste un par un, en testant pour chacun d'eux s'il est plus grand que `maxi`.
3. Si c'est le cas, on a trouvé un candidat potentiel pour le maximum de la liste, on affecte cette valeur à `maxi`, en oubliant celle qui y était stockée, on n'en a plus besoin.
4. Lorsqu'on a parcouru toute la liste, on renvoie `maxi`, c'est le maximum de la liste.

En python, cela donne :

```
maxi = -1
for x in L:
    if x > maxi:
        maxi = x
return maxi
```

On peut apporter de nombreuses variations à cet algorithme, il est possible que la liste d'entier soit en fait une liste chaînée (MPI) ou une structure de données `ocaml` définie dans le sujet, la structure globale est toujours la même.

Saurez-vous écrire une version récursive de l'algorithme qui renvoie le minimum d'une liste non vide, comme le demande le sujet de CCINP filière MP-Info 2025, à la question 14 ?

En langage python, cela donnerait quelque chose comme :

```
def minimum(L):
    if len(L) == 1: # Condition d'arrêt
        return L[0]
    else:
        m = minimum(L[1:])
        return L[0] if L[0] < m else m
```

La recherche du maximum peut sembler simple, ce problème recèle en fait une immense profondeur. Il existe en effet beaucoup de problème en informatique théorique qui consiste à trouver le maximum d'une fonction sur un certain espace, sous certaines conditions, ces problèmes sont regroupés dans ce qu'on appelle **l'optimisation**.

Certains de ces problèmes sont étudiés dans ces annales, nous les détaillons dans la section suivante.

Algorithme glouton

Dans plusieurs sujets, on doit résoudre un problème complexe : il faut prendre plusieurs décisions à la suite, chaque décision ayant un impact sur les suivantes.
Pour trouver la meilleure solution, le sujet vous fait travailler sur plusieurs possibilités :

* la première méthode proposée est souvent la **force brute** qui teste TOUTES les possibilités.
  Une question du sujet vous fait écrire l'algorithme.
  La question qui suit vous fait calculer la complexité de cet algorithme, souvent exponentielle, en $O(2^n)$.
  La troisième question du lot vous fait alors comprendre que son application est inenvisageable : dès que la taille du problème devient un peu grand, le temps de calcul dépasse le nombre d'atomes dans l'univers..

* On vous propose alors une méthode plus efficace, qui consiste à choisir la décision qui semble la meilleure à chaque étape, sans regarder les décisions précédentes. C'est ce qu'on appelle un **algorithme glouton**. Voici quelques exemples :
  - Dans l'épreuve commune du concours Mines-Ponts 2023, le problème est de répartir les mots d'un paragraphe donné sur plusieurs lignes pour pouvoir le justifier le mieux possible. Ce que vous faites d'un simple clic sur votre traitement de texte cache en fait un algorithme sophistiqué.

  - Dans l'épreuve commune de Mines-Ponts 2024, on s'intéresse à un algorithme de cryptographie, on doit déchiffrer un message lettre par lettre, chaque lettre étant associée à une certaine probabilité d'apparition. L'algorithme glouton choisit d'abord la lettre qui a le plus de chances d'apparaître, puis la seconde, etc.

  - C'est le problème du sac à dos qui est étudiée au concours Mines-Ponts 2025, filière MP-Info : Vous devez remplir un sac à dos avec des objets de poids et de valeurs différentes, en maximisant la valeur totale et sans dépasser le poids limite.

Les algorithmes gloutons sont rarement performants. Ils s'exécutent très rapidement, mais on vous propose souvent d'étudier d'autres méthodes de résolution.

Terminons ce paragraphe avec le problème de l'épreuve commune du concours CCINP 2025 :
Un randonneur doit relier un point A à un point B en passant par plusieurs points intermédiaires.
Quel chemin devra-t-il choisir pour que la randonnée globale soit la moins difficile ?
Là encore, l'algorithme glouton ne permettra pas de résoudre le problème.

Si vous comprenez le problème, vous savez que l'algorithme qui permettra au randonneur de trouver la meilleure randonnée est un parcours de graphe.

Les graphes

La théorie des graphes est très riche en problèmes théoriques et algorithmiques, et chacun constitue presque un domaine de recherche à lui seul.

Posséder un culture de ces problèmes vous aidera pour deviner où le sujet veut en venir. Les questions demandant au candidat d'expliquer comment l'algorithme est connecté au problème réel sont fréquentes et souvent mal réussies.

Voici un petit florilège basé uniquement sur les sujets rassemblés dans ces annales :

1. **Colorer les sommets d'un graphe** pour qu'aucune arête ne relie deux sommets de la même couleur. Le sujet MP-Info CCINP 2024 applique ce problème dans un autre contexte :

   Comment constituer des groupes d'étudiants
   pour que deux amis ne soient pas dans le même groupe ?

2. Trouver **un couplage** : un ensemble d'arêtes dans le graphe ne partageant aucun sommet.
   Le sujet de Centrale-Supélec MPI 2025 (partie B) donne une bonne illustration de ce problème : Une casse automobile est modélisée par un plateau, une voiture couvre deux cases du plateau et est modélisée par une arête dans un certain graphe.

   Comment remplir au maximum
   une casse automobile avec des voitures ?

3. Trouver un **flot maximal** dans un graphe.
   Imaginez que votre graphe est un réseau de canalisations, et que le poids de chaque arête représente le débit maximum de la canalisation.
   L'épreuve commune du concours CCINP 2023 propose une application de ce problème : Chaque pixel d'un caractère scanné est codé par le sommet d'un graphe. Suivant la façon dont l'eau s'écoule à travers le caractère, on peut déflouter le caractère.

   Comment identifier un caractère flou
   dans un texte scanné ?

4. Trouver **un arbre couvrant** de poids maximal.
   Intéressons-nous cette fois au sujet du concours CCINP pour la filière MP option Info 2025. On y étudie le réseau internet d'un pays imaginaire, on vous donne les infrastructures existantes et on vous fait construire un algorithme pour construire un réseau de fibre optique sans cycle et avec la plus petite latence.

   Comment construire le réseau de fibre optique
   avec la plus petite latence ?

5. La **recherche du plus court chemin**, et c'est de loin le problème le plus répandu dans ces sujets. Dédions leur la prochaine section de cette introduction.

Parcours de graphe

Un parcours de graphe, ça ressemble à ça :

```
1  def parcours(graphe, sommet_depart):
2      pile = [sommet_depart]
3      visites = {sommet_depart}
4      while pile:
5          sommet = pile.pop()
6          for voisin in graphe[sommet]:
7              if voisin not in visites:
8                  visites.add(voisin)
9                  pile.append(voisin)
10     return visites
```

Bien sûr plusieurs variantes existent mais les ingrédients de base sont les mêmes. Vous devez connaître chaque ligne de cet algorithme, ce qu'elle implique et par quoi on peut la remplacer.

Détaillons un peu ces ingrédients :

- On a une pile (ou une file) (lignes 2, 4, 5, 9) servant à stocker temporairement les prochains sommets à explorer.
- Un tableau (ou un ensemble) (lignes 3, 7, 8) pour marquer les sommets qu'on a déjà visités.
- Une façon de parcourir les voisins d'un sommet (ligne 6), en fonction de la structure de données utilisée pour le graphe, la syntaxe sera différente. Ici, le graphe est décrit par les listes d'adjacences : `graphe[sommet]` est la liste des voisins du sommet `sommet`.

On retrouve ces trois ingrédients dans tous les parcours. (Même dans les versions récursives qu'on vous demandera dans l'épreuve de Mines-Ponts 2024.)

Vous souvenez-vous de la différence entre un parcours en profondeur et un parcours en largeur ?

La seule différence se trouve dans l'ordre dans lequel on parcourt les sommets. Pour donner une métaphore, imaginez devoir trouver la sortie d'un labyrinthe :

- Avec un parcours en profondeur, vous choisissez un chemin et vous le suivez jusqu'au cul de sac, puis vous revenez en arrière pour explorer un autre chemin.
- Avec un parcours en largeur, lorsque vous vous trouvez à une intersection, vous choisissez une direction puis vous avancer jusqu'à la prochaine intersection, vous notez le nombre de chemins qu'il sera possible de prendre à cet endroit, puis vous revenez sur vos pas et faites de même avec les autres directions.

Si vous deviez vraiment marcher le long des couloirs du labyrinthe, vous choisiriez probablement un parcours en profondeur. Pour la plupart des applications, aller d'un sommet à un autre ne coûte aucun effort et c'est généralement un parcours en largeur qui est utilisé.

La plupart du temps, le choix n'est pas important :

- Pour détecter les composantes connexes d'un graphes, ou les groupes de pierres au jeu de go (Centrale MPI 2024), on peut utiliser un parcours en profondeur ou en largeur.
- En revanche, lorsqu'on cherche le plus court chemin dans un graphe, il faut un parcours en largeur. Par exemple, pour trouver la solution d'un jeu de réflexion comme le jeu des randonneurs qui traversent une rivière (MPI CCINP 2023). Ou encore le jeu où il faut faire sortir une voiture d'un parking après avoir déplacé d'autres voitures (MPI Centrale 2024, partie A), on utilise un parcours en largeur.

D'autres variantes élaborées vous seront parfois proposées, par exemple l'algorithme de parcours en largeur multi-sources du sujet de Centrale MPI 2025.

Bien sûr, **l'algorithme de Dijkstra** n'est pas loin, ajoutez des poids sur les arêtes et un simple parcours en largeur ne suffit plus pour obtenir le plus court chemin. Cet algorithme est tellement classique que vous devriez y penser dès que le sujet vous introduit un graphe pondéré.
Revoyez le bien !

L'épreuve Commune de CCINP 2025 vous fera travailler en profondeur cet algorithme ainsi qu'une variante pour accélérer la recherche du plus court chemin.

La version la plus optimale de cet algorithme de Dijkstra utilise une file de priorité pour obtenir plus rapidement le sommet avec la plus petite distance au sommet de départ.

La question du choix d'une structure de données est souvent présente dans ces sujets de concours, la question

Comment stocker ma liste de nombres de la meilleure façon ?

n'admet pas de réponse simple, tout dépend de l'opération que vous effectuez le plus.
Faire une recherche dans cette liste ? Insérer un élément ? Supprimer un élément ?

Allez lire par exemple les parties II et III du sujet de Centrale MP-Info 2023, vous y trouverez plusieurs façons de stocker une bête liste de nombres.

Mis à part vous parler de quelques algorithmes classiques, j'espère avec cette introduction vous avoir donné envie de travailler l'informatique. Les problèmes sont riches et les applications variées.

Nous avons parlé de fond, parlons maintenant de forme.

Quelques conseils de préparation

Lors de votre préparation à l'écrit, tapez et **compilez vos codes**, ne vous contentez pas d'écrire des algorithmes sur papier !

Servez-vous de **l'intelligence artificielle**, elle peut vous permettre de rédiger des tests, ou des fonctions d'affichage pour voir le résultat de vos programmes.

- Écrivez une première version de votre code avant de décrire le comportement que vous souhaitez, elle vous aidera à creuser ou à expliquer une ligne spécifique.
- Expliquez-lui l'algorithme que vous avez en tête, cela vous forcera à formuler les idées principales le plus clairement possible.
  (Le premier outil qu'ont utilisé les informaticiens pour chercher les erreurs dans leur code est un canard en plastique à qui ils essaient d'expliquer leur algorithme)
- Exigez de comprendre tout ce qu'elle vous donne. L'IA pourra toujours se tromper, même sur des algorithmes simples, elle peut aussi utiliser des fonctions ou des structures de données qui ne sont pas au programme de l'écrit.

Pour rédiger plus vite, entraînez-vous à rédiger des algorithmes sur papier. Lorsque vous travaillez un sujet, prenez bien le temps pour le comprendre, découpez votre travail en plusieurs sessions.
Une fois que vous avez terminé, lancez le chronomètre et rédigez le sujet en entier. Cette pratique vous donnera un regard différent sur le sujet, reliera les parties qui pouvaient sembler éloignées.

À l'écrit, vous devrez coupler les deux compétences : trouver un sens global au sujet en répondant aux questions et écrire efficacement, pendant la préparation séparez les deux.

Quelques conseils de rédaction

La majorité des points de cette épreuve concerne l'écriture d'algorithmes, les correcteurs savent déchiffrer les idées derrière vos codes, ne perdez pas de temps à expliquer tous les détails avec des commentaires.
N'alourdissez pas votre code avec des caractères pour indiquer les indentations, tracez proprement des traits discrets verticaux et respectez-les.

Les rapports de jurys conseillent d'écrire très peu de commentaires, mais d'accompagner votre algorithme d'une ou deux phrases pour expliquer les idées principales. C'est ce que j'ai essayé de faire en rédigeant ce livre.

Conservez les signatures des fonctions au fur et à mesure du sujet, gardez-les dans un coin de votre brouillon. Ce n'est pas rare dans un sujet d'utiliser dans une question une fonction définie une quinzaine de questions plus tôt.

Sur ces mots, profitez de cette plongée dans les sujets, et bonne chance pour l'écrit !

Première partie

Épreuves communes

Chapitre 2

CCINP & E3A : MP, PC, PSI

SESSION 2023 **PSI5IN**

ÉPREUVE MUTUALISÉE AVEC E3A-POLYTECH

ÉPREUVE SPÉCIFIQUE - FILIÈRE PSI

INFORMATIQUE

Durée : 3 heures

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.*

Les calculatrices sont interdites.

Le sujet est composé de trois parties.

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 et Annexe : 16 pages
Document Réponse (DR) : 12 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 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(p,q,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
```

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`.

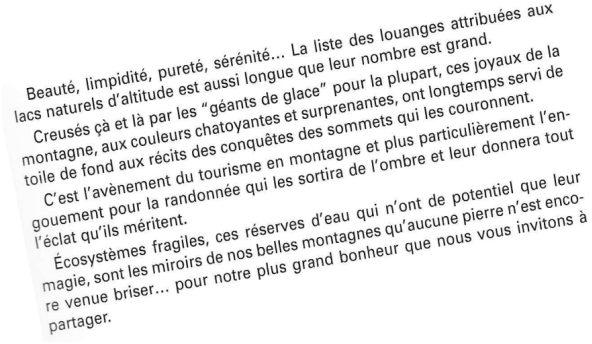
Q3. Proposer une fonction `binarisation(imgG:array, seuil:int)->array` qui convertit une image en niveau de gris en image en noir et blanc en imposant une valeur 255 pour tout pixel de valeur strictement supérieure au seuil.

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).

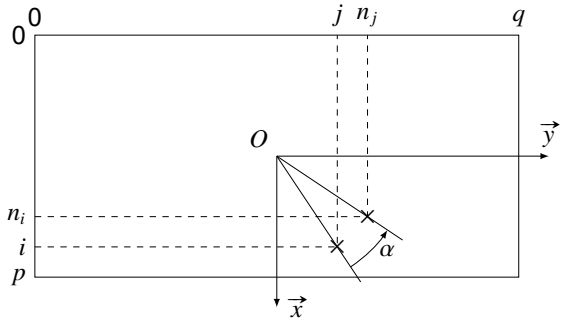


Figure 2 - Paramétrage de l'image pour la rotation

Naïvement, on pourrait penser que pour réaliser la rotation, il suffit de parcourir chaque pixel de l'image intiale 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.

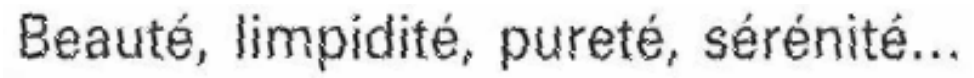


Figure 3 - Rotation naïve de l'image intiale

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**).

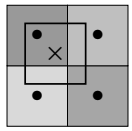


Figure 4 - Illustration du pixel trouvé entouré des 4 pixels voisins dans l'image intiale

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 $x0$ 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)->int` 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.

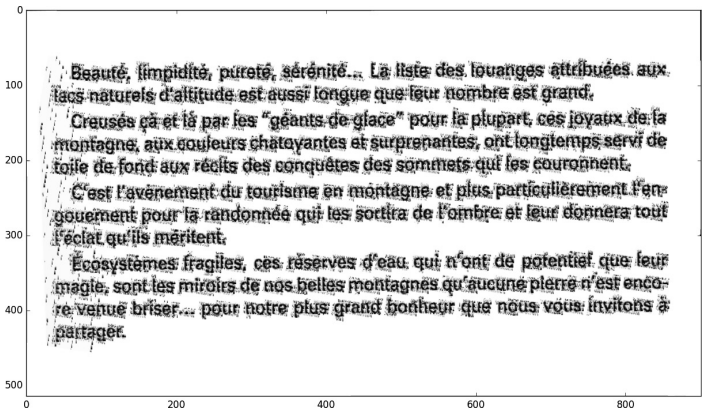


Figure 5 - Problème de pixellisation lors de la rotation

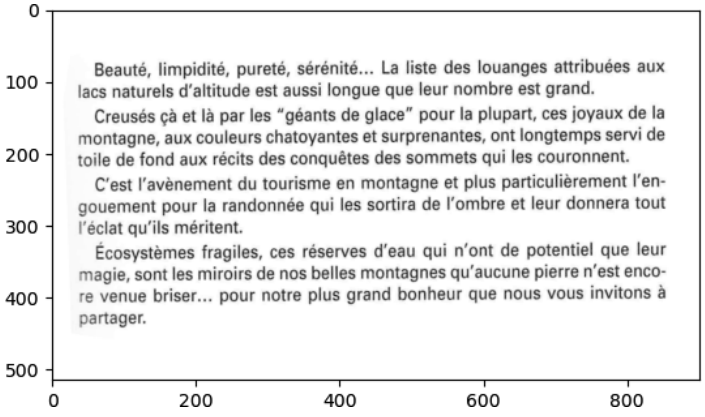


Figure 6 - Figure attendue après la rotation

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**.

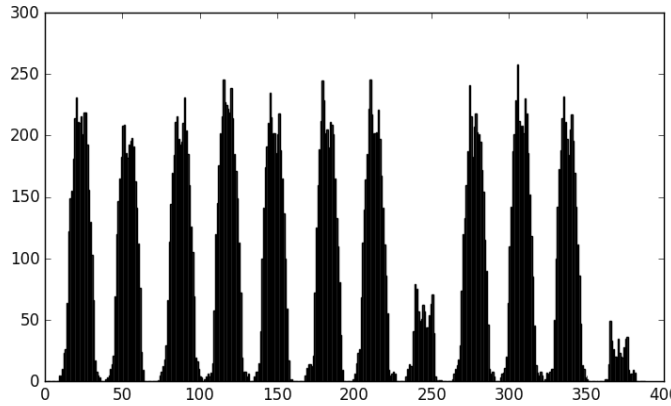


Figure 7 - Histogramme de détection des lignes

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.

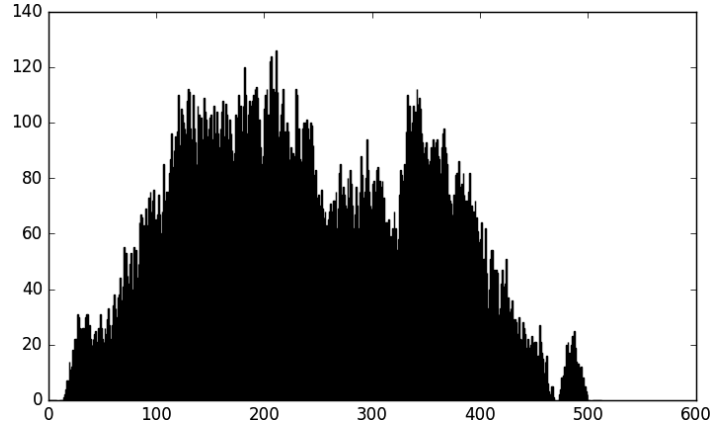


Figure 8 - Histogramme de détection des lignes sur la figure mal orientée

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 $c = \dfrac{a+b}{2}$;
- 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

II.3 - 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 (3x3) constituée de 9 pixels en niveau de gris (pixels compris entre 0 (noir) et 255 (blanc)).

1 : 0	2 : 210	3 : 190
4 : 20	5 : 100	6 : 200
7 : 10	8 : 5	9 : 255

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.

Q11. Représenter le graphe correspondant à l'image de (3x3) 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)`.

Q12. 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 $\mathbf{R}$. 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.

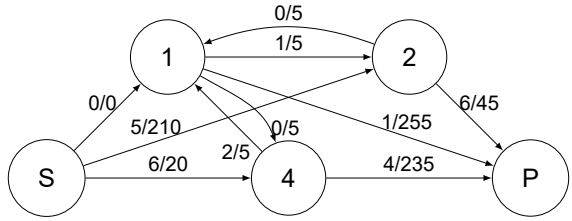


Figure 11 - Exemple d'extrait de graphe avec flot

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**.

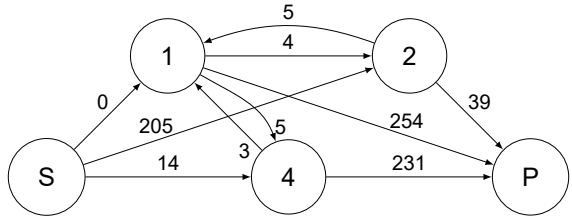


Figure 12 - Exemple de graphe résiduel

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 où la somme des étiquettes du graphe résiduel en parcourant les arêtes le constituant est minimal et 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
définir un chemin C de S à P dans le graphe résiduel de longueur minimale
tant qu'il existe un chemin C de S à P dans le graphe résiduel faire
   prendre un chemin C de longueur minimale
   a = min(r(e)| e dans C)
   pour tout e dans C faire
      f(e) = f(e) + a
   fin pour
   mettre à jour le graphe résiduel
fin tant que
```

Q13. 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.

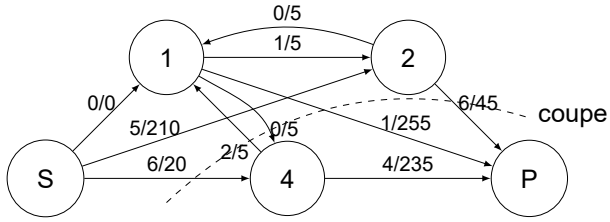


Figure 13 - Coupe dans un graphe

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.

Q14. 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**.

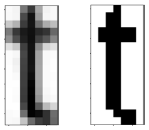


Figure 14 - Caractère scanné et caractère après traitement par la fonction `graph_cut`

Q15. Indiquer pour cette image de la **figure 14** à quoi correspondent les ensembles A et B. Analyser le résultat obtenu.

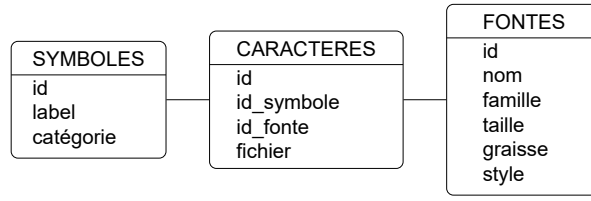
Partie III - Détermination des caractères

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).

Q16. 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.

Q17. Écrire une requête SQL permettant d'extraire tous les noms de fichiers des caractères qui correspondent au symbole de label `"A"`.

Q18. 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 : `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`.

Q19. 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`.

Q20. 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 $d = \sqrt{\Sigma_{i,j}(p_{ij} - q_{ij})^2}$.

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.

Q21. Écrire une fonction `distance(im1:array, im2:array)->float` qui calcule la distance entre les deux images `im1` et `im2` supposées de même dimension.

Q22. 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.

Q23. 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**.

Q24. Compléter les 3 zones manquantes dans cet algorithme.

Q25. 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 K ne dépassera pas 5.

Q26. 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,"`

Q27. Commenter les résultats obtenus.

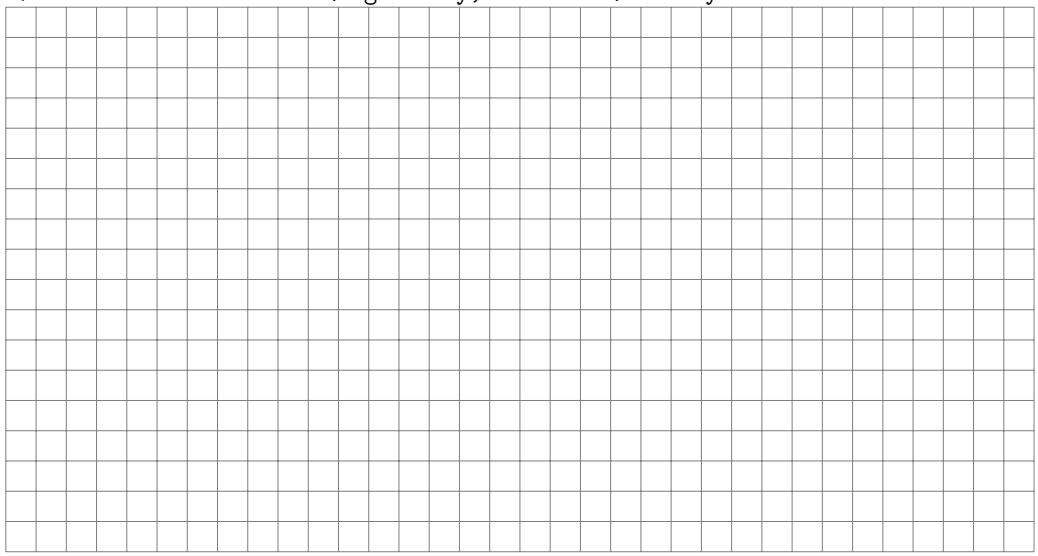
ANNEXE

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

Q3 - Fonction `binarisation(imgG:array, seuil:int)->array`



Q4 - Choix de la fonction `bilineaire(im:array, x:float, y:float)->int`

```
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)
```

```
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)
```

```
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)
```

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

Q5 - Fonction `rotation(im:array, angle:float)->array`

```
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
```

Q6 - Étude de la fonction `lineaire`

Q7 - Fonction `histo_lignes(im:array)->list`

Q8 - Fonction `detecter_lignes(liste:list)->list`

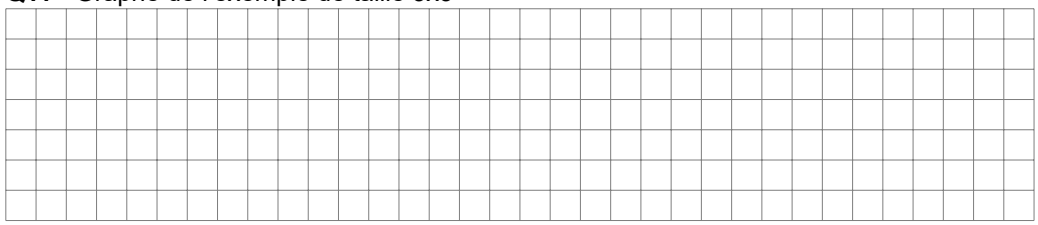
```
def detecter_lignes(liste:list)->list:
    lignes = []
    i = 0
    deb = -1 #contient -1 tant qu'on parcourt des lignes de pixel blanc
    while ..................  :
        #début d'une suite de lignes contenant des pixels noirs
        if ........................:
            deb = ...........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

Q9 - Justification de la terminaison et nom de la méthode. Nombre d'itérations nécessaires

Q10 - Fonction `rotation_auto(im:array, a:float, b:float)->array`

```
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)
```

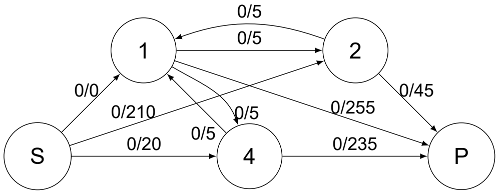
Q11 - Graphe de l'exemple de taille 3x3

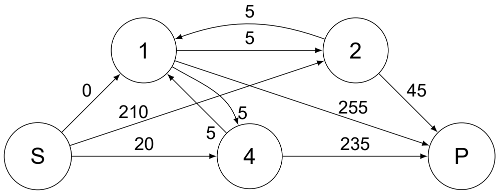


Q12 - Compléter la partie supérieure de la matrice de capacités

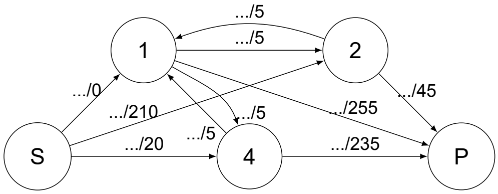
	S	1	2	3	4	5	6	7	8	9	P
S											
1	-										
2	-	-									
3	-	-	-								
4	-	-	-	-							
5	-	-	-	-	-						
6	-	-	-	-	-	-					
7	-	-	-	-	-	-	-				
8	-	-	-	-	-	-	-	-			
9	-	-	-	-	-	-	-	-	-		
P	-	-	-	-	-	-	-	-	-	-	

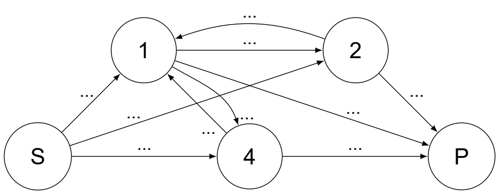
Q13 - Compléter les graphes de flot à gauche et résiduel à droite. Pour chaque étape, préciser le chemin choisi et la valeur de l'augmentation du flot



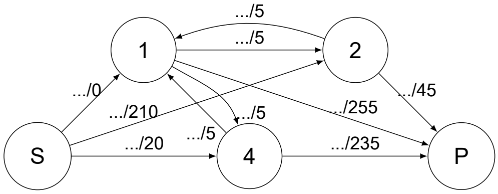


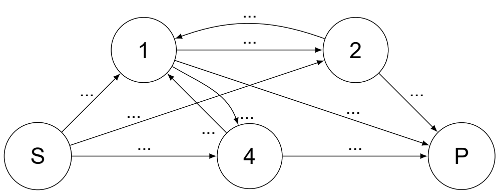
Chemin choisi :



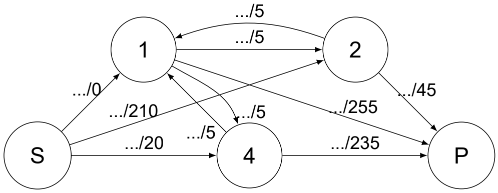


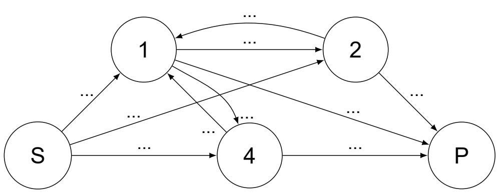
Chemin choisi :





Chemin choisi :





Q14 - Ensembles A et B. Flot maximal. Coupe choisie et capacité de coupe

Q15 - Correspondance ensembles A et B. Analyse

Q22 - Fonction `calcul_distances(carac_ref:dict, carac_test:array)->dict`

Q23 - Méthode de tri performante envisageable et complexité temporelle

Q24 - Compléter les 3 zones manquantes

```
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
```

Partie I - Acquisition d'un document

• **Q1.** Il faut 8 bits pour représenter un entier entre 0 et 255.

Au brouillon

Écrivez le tableau de conversion :

Nombre de bits	1	2	3	4	5	6	7	8
Nombres représentables	2	4	8	16	32	64	128	256

Si un pouce est environ égal à 2.5 centimètres, alors :

— 21 cm est environ égal à 21/2.5 pouces.

— 29.7 cm est environ égal à 29.7/2.5 pouces.

La résolution de l'image est de 300 pixels par pouce, pour obtenir le nombre de pixels, il faut donc multiplier chacun des nombres ci-dessus par 300, cela correspond donc à

$$300^2 \times 21 \times 29.7/2.5^2 \text{ pixels.}$$

Comme chaque pixel est codé sur 3 couleurs, et que chaque couleur est codée sur 8 bits, pour obtenir le nombre de bits, il faut multiplier le nombre de pixels par 3 et par 8, cela correspond à

$$300^2 \times 21 \times 29.7 \times 3 \times 8/2.5^2 \text{ bits.}$$

Remarque

Pour obtenir une estimation, on pouvait travailler en ordre de grandeur :

$$300^2 \sim 10^5, \qquad 21 \sim 2 \times 10^1, \qquad 29.7 \sim 3 \times 10^1, \qquad 2.5^2 \sim 6$$

donc

$$300^2 \times 21 \times 29.7/2.5^2 \sim 10^7.$$

Reste à multiplier par $3 \times 8 \sim 3 \times 10^1$ on obtient finalement environ 10^8 bits, soit 100 Mbits ou une dizaine de Mega octets, ce qui semble raisonnable pour une grosse image.

Rapport de Jury

Les correcteurs rapportent une question mal traitée, prenez le temps de soigner la première question du sujet. Elle influe grandement sur le regard que le correcteur porte sur votre copie.

Remarque

Le code fourni dans le sujet contient une erreur, il faut remplacer p par n_0 et q par n_1 dans la deuxième ligne de la fonction.
L'avant dernière ligne aurait également dû s'écrire

`img[i][j]= int(val)` au lieu de `img[i,j]= int(val)`.

Lorsque cela se produit dans un sujet, le jury est indulgent, ne vous laissez pas perturber et continuez.

- **Q2.** Cette fonction contient deux boucles imbriquées la première s'exécute q fois, la seconde p fois. Le nombre de tours de boucle est donc $q \times p$, qui sont les dimensions de l'image. De plus les 3 instructions qui sont exécutées à chaque tour de boucle sont des affectations ou des opérations élémentaires, elles s'effectuent donc en temps constant.

On ne nous donne pas les complexités des deux fonctions `dimension` et `initialise`, on peut supposer qu'elles sont aussi en $O(p \times q)$.

La complexité de la fonction `inversion` est donc en $O\big(\text{nombre de lignes} \times \text{nombre de colonnes}\big)$.

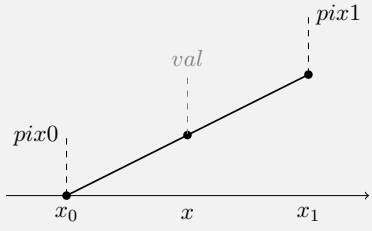
- **Q3.** Cette fonction initialise un nouveau tableau blanc, parcourt l'image donnée en paramètre et change un pixel en noir si celui-ci dépasse le seuil :

```
1  def binarisation(imgG, seuil):
2      n_0, n_1, _ = dimension(imgG)
3      imgNB = initialise(n_0, n_1, 0)
4      for i in range(n_0):
5          for j in range(n_1):
6              gris = imgG[i][j]
7              if gris > seuil:
8                  imgNB[i][j] = 255
9      return imgNB
```

Partie II - Reconnaissance du document

Au brouillon

Pour bien comprendre les notations de la question, on peut faire un dessin au brouillon :



- **Q4.** On suit les indications de l'énoncé :
  — La première interpolation se fait sur la première ligne donc entre les coordonnées

$$(x_0, y_0) \text{ et } (x_0, y_0 + 1).$$

  — La seconde interpolation se fait sur la deuxième ligne donc entre les coordonnées

$$(x_1, y_0) \text{ et } (x_1, y_0 + 1).$$

  C'est donc la proposition en haut à droite qui convient.

- **Q5.**

Remarque

Le programme proposé dans le document réponse invite à utiliser n_i et n_j comme variables de boucles, qui représentent les coordonnées des points après rotation dans la figure de l'énoncé. Faisons confiance à l'énoncé, et implémentons l'algorithme pour qu'il calcule la formule donnée.

Il faut bien penser que les fonctions `np.cos` et `np.sin` prennent des arguments en radians.

```
def rotation(im, angle):
    p,q, _ = dimension(im)
    imr = initialise(p, q, 255)
    angr = np.radians(angle)
    matR = np.array([[np.cos(angr), np.sin(angr)],
                     [-np.sin(angr), np.cos(angr)]])
    for ni in range(p):
        for nj in range(q):
            x, y = prod_matrice_vecteur(matR, np.array([ni-p//2, nj-q//2]))
            x = x + p//2
            y = y + q//2
            if 0 <= x < p and 0 <= y < q:
                imr[ni][nj] = im[int(x)][int(y)]
    return imr
```

• **Q6.** Comme chaque pixel de l'image est codé par un (ou trois) entier(s) entre 0 et 255, se restreindre à 8 bits permet d'économiser de l'espace mémoire. En python, les nombres flottants sont codés par défaut sur 64 bits, et les `int` sur 32 bits.

Si les entiers sont non signés, on ne peut pas représenter de nombres négatifs donc le résultat de la soustraction $18 - 23$ ne peut pas être -5. Le résultat renvoyé par python serait plutôt $(18 - 23) \mod 256 = 251$.

Rapport de Jury

Peu de candidats savent dire ce qui se passe quand on essaie d'entrer un nombre négatif dans un type non signé, la plupart pensent que le nombre est remplacé par sa valeur absolue.

• **Q7.** Il faut créer un tableau, chaque case de ce tableau représente une ligne de l'image et contient le nombre de pixels noirs de cette ligne.

Remarque

Commencez par rédiger la boucle intérieure : sur la i-ième ligne, comment compter le nombre de pixels noirs ? :

```
sum = 0
for j in range(q):
    if im[i][j] == 0:
        sum += 1
```

Ensuite, il vous suffit d'ajouter la boucle extérieure. Vous placerez ainsi naturellement la remise à zéro du compteur. C'est sur ce point que beaucoup de candidats se sont trompés.

La fonction pourrait s'écrire :

```
1  def histo_lignes(im):
2      p, q, _ = dimension(im)
3      result = []
4      for i in range(p):
5          sum = 0
6          for j in range(q):
7              if im[i][j] == 0:
8                  sum += 1
9          result.append(sum)
10     return result
```

• **Q8.**

Au brouillon

Il faut bien faire attention aux indices à insérer dans la liste : ils doivent correspondre à des positions de 0.
Il faut également faire attention à ne pas dépasser la longueur de la liste, surtout si on écrit `liste[i+1]`.

La variable `deb` permet de dire si on se trouve ou non à l'intérieur d'un bloc.

```
def detecter_lignes(liste):
    lignes = []
    i = 0
    deb = -1
    while i < len(liste):
        if deb == -1 and liste[i] != 0:
            deb = i
        if deb != -1 and liste[i] == 0:
            lignes.append([deb, i])
            deb = -1
        i += 1
    return lignes
```

• **Q9.** La longueur de l'intervalle `[a, b]` est réduite de moitié à chaque étape de l'algorithme. La longueur de cet intervalle passera donc sous n'importe quel epsilon donné après un certain rang, ce qui terminera l'algorithme.

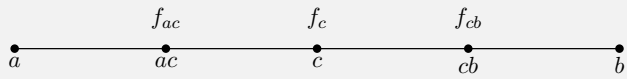
C'est un algorithme par **dichotomie**, après n étapes, la longueur de l'intervalle est $(b-a)/2^n$, l'algorithme s'arrêtera dès que $\frac{b-a}{2^n} < \varepsilon$.

$$\begin{aligned} & \frac{b-a}{2^n} < \varepsilon \\ \iff \quad & 2^n > \frac{b-a}{\varepsilon} \\ \iff \quad & n \ln 2 > \ln \frac{b-a}{\varepsilon} \\ \iff \quad & n > \frac{\ln \frac{b-a}{\varepsilon}}{\ln 2} = \log_2 \frac{b-a}{\varepsilon} \end{aligned}$$

• **Q10.**

Au brouillon

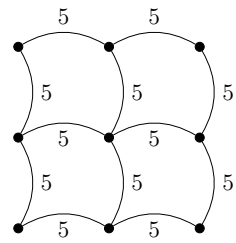
N'hésitez pas à faire un dessin pour bien placer les notations :



Cela vous aidera pour rédiger le programme.

```
def rotation_auto(im, a , b):
    c = (a+b)/2
    fc = nb_zeros(im, c)
    while b-a > 0.1 :
        ac = (a+c)/2
        fac = nb_zeros(im, ac)
        cb = (c+b)/2
        fcb = nb_zeros(im, cb)
        maxi = max(fac, fc, fcb)
        if maxi == fac:
            b = c
            c = ac
            fc = fac
        elif maxi == fc:
            a = ac
            b = cb
        elif maxi == fcb:
            a = c
            c = cb
            fc = fcb
    return rotation(im, (a+b)/2)
```

• **Q11.** Le graphe peut se représenter de la manière suivante, chaque arête a un poids 5 :



• **Q12.** On remplit le tableau de la manière suivante :

	S	1	2	3	4	5	6	7	8	9	P
S	0	0	210	190	20	100	200	10	5	255	0
1		0	5	5	5	5	5	5	5	5	255
2			0	5	5	5	5	5	5	5	45
3				0	5	5	5	5	5	5	65
4					0	5	5	5	5	5	235
5						0	5	5	5	5	155
6							0	5	5	5	55
7								0	5	5	245
8									0	5	250
9										0	0
P											0

• **Q13.**

Remarque

L'algorithme fourni dans l'énoncé est ambigu, on nous demande de trouver un plus court chemin, c'est-à-dire un chemin pour lequel la somme des étiquettes du graphe résiduel en parcourant les arêtes le constituant est minimal et comportant le moins d'arêtes.
Cette formulation peut vouloir dire deux choses :

A Un chemin de S à P qui minimise la somme des étiquettes des arcs, en cas d'égalité entre plusieurs chemins, on choisit le chemin comportant le moins d'arêtes.

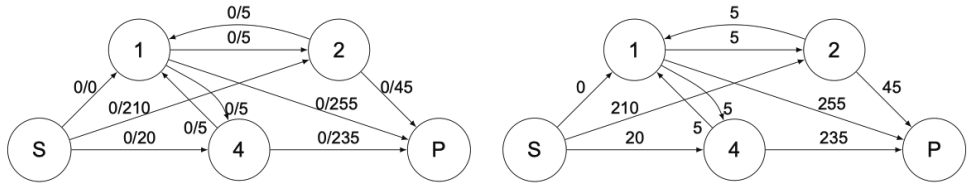
B Un chemin de S à P qui minimise le nombre d'arêtes, en cas d'égalité entre plusieurs chemins, on choisit le chemin qui minimise la somme des étiquettes.

Voici ce qu'en dit le rapport de Jury :

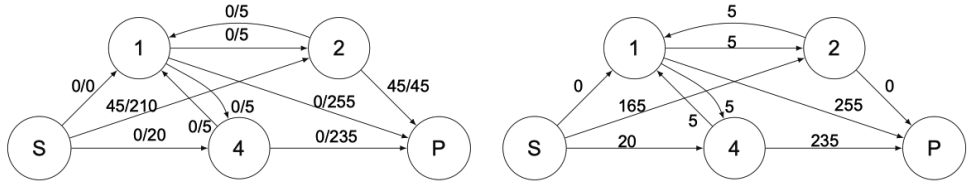
> Cette ambiguïté est présente dans l'énoncé, le jury accepte les deux raisonnements pour cette question et la suivante.
>
> La version **A** demandait cependant plus d'étapes que ne le prévoyait le document réponse.

Le rapport prétend que la version **A** conduit au même résultat, mais c'est faux.
Votre premier chemin sera $S \to 4 \to 1 \to 2 \to P$. Votre deuxième et troisième chemin seront $S \to 4 \to P$ ou $S \to 2 \to P$, peu importe l'ordre. Vous pourrez encore choisir un chemin $S \to 2 \to 1 \to P$ mais votre flot ne sera jamais maximal car vous allez saturer l'arête $4 \to 1$.
Il fallait donc comprendre la version **B** pour répondre correctement à la question.

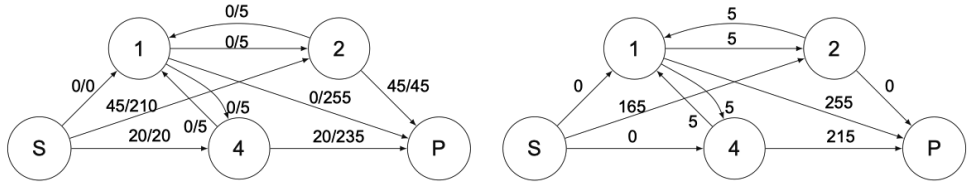
Voici dans le cas **B** les chemins donnés par l'algorithme fourni : On peut inverser le premier et le deuxième chemin, ils ont la même somme d'étiquette et le même nombre de sommets.



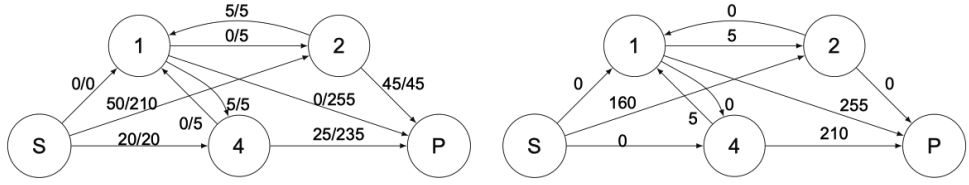
Chemin choisi : S - 2 - P, capacité maximale : 45



Chemin choisi : S - 4 - P, capacité maximale : 20



Chemin choisi : S - 2 - 1 - 4 - P, capacité maximale : 5



• **Q14.** La coupe minimale est celle qui sépare les ensembles $A = \{S, 2\}$ et $B = \{1, 4, P\}$. La capacité de cette coupe correspond à la somme des étiquettes des arêtes qui séparent A et B :

$$S \to 4,\ 2 \to 1 \text{ et } 2 \to P : 20 + 5 + 45 = 70.$$

Le flot obtenu est la somme des flots partant de S, c'est-à-dire $50 + 20 = 70$.
Ces deux nombres sont bien égaux, donc le flot est maximal.

Remarque

C'est cohérent avec le dessin de la figure 10 dont notre graphe est issu : les pixels 1 et 4 seront probablement de la même couleur, et le pixel 2 sera d'une couleur différente.

• **Q15.** Dans la figure 14, l'ensemble A correspond aux pixels blancs, et l'ensemble B aux pixels noirs.

Partie III - Détermination des caractères

• **Q16.** C'est une requête simple, il faut bien rédiger les conditions :

```
SELECT id FROM FONTES
WHERE nom = 'Zurich' AND style = 'romain' AND taille BETWEEN 10 AND 16;
```

Rapport de Jury

Le jury semble préférer une double comparaison à l'utilisation du `BETWEEN`, qui est souvent mal maîtrisé. La requête donnerait :

```
SELECT id FROM FONTES
WHERE nom = 'Zurich' AND style = 'romain' AND taille >= 10 AND taille <= 16;
```

Choisissez la version que vous préférez.

• **Q17.** Il faut réaliser une jointure des deux tables

```
SELECT C.fichier FROM CARACTERES C
JOIN SYMBOLES S ON C.id_symbole = S.id
WHERE S.label = 'A';
```

Rapport de Jury

Prêtez bien attention aux noms des identifiants qui réalisent la jointure !

Remarque

L'utilisation des mots-clés `AS` est optionnelle.

• **Q18.** Requête plus difficile avec un `COUNT`

```
SELECT S.label, COUNT(C.id) AS nombre_caracteres
FROM CARACTERES C
    JOIN FONTES F ON C.id_fonte = F.id
    JOIN SYMBOLES S ON C.id_symbole = S.id
WHERE F.nom = 'Zurich'
    AND F.style = 'romain'
    AND F.taille BETWEEN 10 AND 16
GROUP BY S.label;
```

Rapport de Jury

Le `GROUP BY` se place après les `WHERE` et avant le `SELECT`.

• **Q19.** Examinons ces variables :

- `car` contient le tableau `[nomFonte, nomCatégorie+taillePolice, idSymbole+".png"]`
- `num` contient la variable `idSymbole`
- `var` contient la variable `nomCatégorie` à condition que la taille de la police ait exactement 2 chiffres.
- `ind` contient l'indice de la catégorie dans le tableau `categories`

Rapport de Jury

Si vous sentez que vous manquez de temps, ne vous précipitez pas, mieux vaut assurer les points à cette question que bâcler les 3 suivantes.

Si on appelle la fonction avec cet argument, la fonction renvoie `"K"`.

• **Q20.** On parcourt toute la liste des noms de fichiers, si c'est un symbole qu'on n'a jamais rencontré, on crée une entrée dans le dictionnaire qui est un tableau vide. On l'ajoute ensuite à la liste des images correspondant à ce symbole.

```
def lire_donnees_ref(fichiers_car_ref):
    result = {}
    for fichier in fichiers_car_ref:
        symbole = lire_symbole_fichier(fichier)
        tableau = imread(fichier)
        if symbole not in result:
            result[symbole] = []
        result[symbole].append(tableau)
    return result
```

Rapport de Jury

Le point critique était ici l'initialisation du dictionnaire si la clé n'est pas présente.

• **Q21.** On parcourt chaque pixel des deux images et on met à jour une variable qui additionne toutes les distances élémentaires progressivement.

```
def distance(im1, im2):
    sum = 0
    n, p = size(im1)
    for i in range(n):
        for j in range(p):
            sum += (im1[i][j] - im2[i][j])**2
    return sqrt(sum)
```

• **Q22.** Chaque valeur du dictionnaire sera un tableau de distances.

```
def calcul_distances(carac_ref, carac_test):
    dist = {}
    for symbole_array in carac_ref:
        dist[symbole_array] = []
        for image in symbole_array:
            dist[symbole_array].append(distance(carac_test, image))
    return dist
```

• **Q23.** On peut lister plusieurs algorithmes de tri performants :

- Le tri rapide, complexité en $O(n \ln n)$ ou $O(n^2)$ dans le pire des cas
- Le tri fusion, complexité en $O(n \ln n)$ (mais nécessite un tableau auxiliaire)
- Le tri par tas, complexité en $O(n \ln n)$ (mais ne préserve pas l'ordre initial des éléments égaux)

Rapport de Jury

Vous devez connaître les noms et complexités des algorithmes de tri de base.

• **Q24.**

Au brouillon

Commencez par réécrire l'algorithme de tri par insertion puis adaptez-le pour compléter le document en annexe.

Remarque

Le tri par insertion est un algorithme qui explore progressivement la liste et trie les éléments un par un.

- Le premier élément reste où il est
- On regarde le deuxième élément :
  - S'il est bien placé par rapport au premier, on le laisse.
  - Sinon on le place en première position
- On regarde le troisième élément :
  - S'il est bien placé par rapport aux deux premiers, on le laisse.
  - Sinon on le place au bon endroit parmi les deux premiers
- ...

Remarque

Voici comment implémenter cet algorithme :

```
def tri_insertion(liste):
    for i in range(1, len(liste)):
        element_courant = liste[i]
        j = i - 1
        while j >= 0 and liste[j] > element_courant:
            liste[j + 1] = liste[j] # On décale les éléments
            j -= 1
        liste[j + 1] = element_courant # On insère l'élément au bon endroit
    return liste
```

C'est le tri que vous réalisez lorsque vous classez les cartes à jouer dans votre main. Entraînez-vous à écrire les algorithmes de tri ! Ils doivent sortir sans hésitation.

Dans le code qui suit, le tableau `voisins` est toujours trié par ordre croissant des distances. On compare `d[j]` avec `voisins[-1][0]` qui est la plus grande distance dans le tableau.

La boucle `while` permet de décaler les éléments du tableau `voisins` pour faire de la place à `d[j]` comme dans l'algorithme de tri par insertion.

```
def Kvoisins(distances, K) :
    voisins = [(float("inf"), "") for k in range(K)]
    for lettre in distances:
        d = distances[lettre]
        for j in range(len(d)):
            if d[j] < voisins[-1][0]:
                k = len(voisins) - 1
                while k > 0 and d[j] < voisins[k - 1][0]:
                    voisins[k] = voisins[k - 1]
                    k = k - 1
                voisins[k] = (d[j], lettre)
    return voisins
```

- **Q25.** La complexité de cet algorithme est en $O(nK)$. Avec un algorithme de tri classique, la complexité est en $O(n \ln n)$, si on sait que K est très petit par rapport à n, on peut donc se limiter à trier les K plus petites distances.

• **Q26.** On compte, pour chaque symbole, le nombre de fois qu'il apparaît parmi les K plus petites distances :

```
def symbole_majoritaire(voisins):

    compteur = {}
    for _, symbole in voisins:
        if symbole in compteur:
            compteur[symbole] += 1
        else:
            compteur[symbole] = 1
    symbole_majoritaire = max(compteur, key=compteur.get)

    return symbole_majoritaire
```

• **Q27.** Pour obtenir de bons résultats, il est important d'avoir beaucoup d'images pour chaque symbole, le mieux étant de maximiser le nombre de polices différentes.

Si $K = 1$, on prend seulement la plus petite distance, ce qui n'est pas représentatif. Une valeur de $K = 4$ donne de meilleurs résultats.

Le résultat optimal est donné par l'algorithme qui exploite le plus de données.

Fin

SESSION 2024 **PSI5IN**

ÉPREUVE MUTUALISÉE AVEC E3A-POLYTECH

ÉPREUVE SPÉCIFIQUE - FILIÈRE PSI

INFORMATIQUE

Durée : 3 heures

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, bleu clair ou turquoise, 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.*

Les calculatrices sont interdites.

Le sujet est composé de trois parties.

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 de manière raisonnable.

Énoncé et Annexe : 16 pages
Document Réponse : 12 pages

Seul le Document Réponse doit être rendu dans son intégralité (le QR Code doit être collé sur la première page de ce Document Réponse).