CCINP 2023

1)
La taille nécessaire d'un mot binaire pour stocker un entier entre 0 et 255 est de 8 bits (2^8 valeurs).
La largeur en pixel L_p est donnée par R * L_{cm}. De même pour la hauteur H_p. Il faut convertir la résolution de 300 ppp à la résolution en p/cm.
$L_p \approx \frac{20*300}{2} \approx 3000$ pixels
$H_p \approx \frac{30*300}{2} \approx 4500$ pixels
Donc l'image est une 3000 * 4500 pixels environ.
Ne pas oublier, sans calculettes -> ordre de grandeurs

2)
On remarque qu'il y a pas mal de coquilles dans le sujet, notamment ici n0 et n1 se transforment en p, q.
La complexité est en $O(n0 * n1)$.
Attention à bien exprimer les complexités en fonction des paramètres, dire que la complexité est juste quadratique ou linéaire n'est pas suffisant, car elle est quadratique en fonction de la largeur ou hauteur des images, mais linéaire en fonction du nombre de pixels.

3)
```
def binarisation(imgG:array,seuil:int) -> array :
    n0, n1, _  = dimension(imgC)
    img = initalise(n0,n1,0)
    for i in range(n0) :
        for j in range(n1) :
            g = imgC[i][j]
            if g > seuil :
                img[i][j] = 255
            else :
                img[i][j] = 0
    return img
```

4)
La fonction bilinéaire à prendre est la fonction en haut à droite.
En effet, la variable a est la moyenne pondérée par $y0$ et $y1$ au point d'ordonnée y par *linéaire* des deux pixels de gauche, et b est la moyenne pondérée pour les deux pixels de droite. Ensuite, c est la moyenne pondérée de ces deux points.

5)
```
def rotation(im:array,angle:float) -> array :
    p, q, _ = dimension(im)
    imr = initialise(p,q,255)
    angr = -angle / 180 * pi
    matR = [[cos(angr),-sin(angr)],[sin(angr),cos(angr)]]
    for ni in range(p) :
        for nj in range(g) :
            x , y = prod_matrice_vecteur(MatR,[ni-p//2,nj-q//2]
            x = x + p//2
            y = y + q//2
            if x >= 0 and x <= p-1 and y >= 0 and y <= q-1  :
                imr[ni][nj] = bilineaire(im,x,y)
    return imr
```

6)
Contraindre la sortie de linéaire à un entier non signé permettrait un gain de place en mémoire par rapport à un flottant.
$18-23 \equiv -5[256] \equiv 251[256]$ Il y a dépassement, et le résultat de 18-23 est corrompu, et vaudra 251.

7)
```
def histo_lignes(im:array) -> list :
    rep = []
    n0, n1, _= dimension(im)
    for i in range (n0) :
        compte = 0
        for j in range(n1) :
            if im[i][j] != 0 :
                compte += 1
        rep.append(compte)
    return rep
```

8)

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

9)

Le nom de l'algorithme est la dichotomie. À chaque tour de boucle, la taille de l'intervalle [a,b] est divisé par deux. Donc, au bout d'un moment, cette taille sera plus petite que ϵ.

Si on note N le nombre d'itérations, il n'y a pas arrêt à la N-1ème boucle, donc :

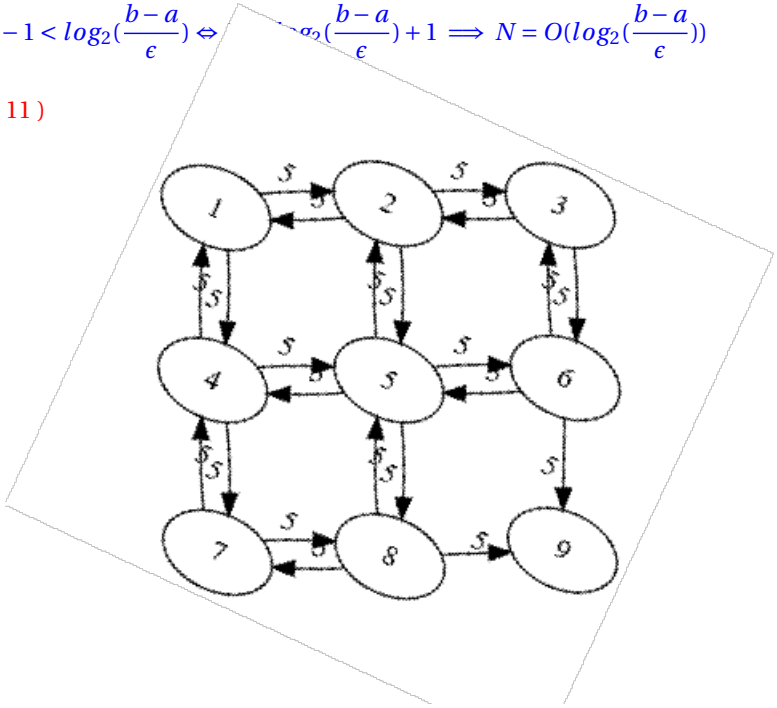
$$\frac{b-a}{2^{N-1}} > \epsilon \Leftrightarrow 2^{N-1} < \frac{b-a}{\epsilon} \Leftrightarrow N-1 < log_2(\frac{b-a}{\epsilon}) \Leftrightarrow \dots log_2(\frac{b-a}{\epsilon}) + 1 \Longrightarrow N = O(log_2(\frac{b-a}{\epsilon}))$$

10)

Seules les lignes à remplir sont notées dans la correction.

```
[...]
    ac = (a+c)/2
    fac = nb_zeros(im,ac)
    cb = (c+b)/2
    fcb = nb_zeros(im,ac)
    [...]
    if fac == maxi
    [...]
    elif fc == maxi
    [...]
    else :
        a = c
        c = cb
        fc = fcb
```

11)

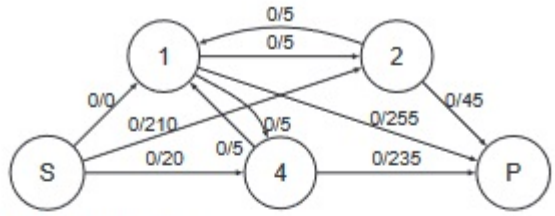


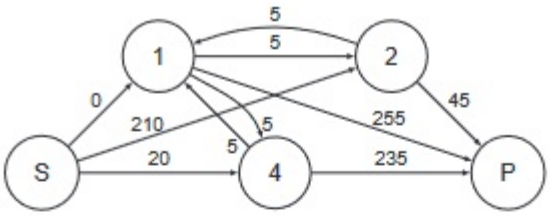
12)

Q12 - Compléter la partie supérieure de la matrice de capacités

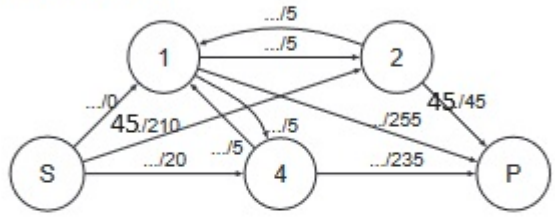
	S	1	2	3	4	5	6	7	8	9	P
S	0	0	210	190	20	100	200	10	5	255	
1	-	0	5	0	5	0	0	0	0	0	255
2	-	-	0	5	0	5	0	0	0	0	45
3	-	-	-	0	0	0	5	0	0	0	65
4	-	-	-	-	0	5	0	5	0	0	235
5	-	-	-	-	-	0	5	0	5	0	155
6	-	-	-	-	-	-	0	0	0	5	55
7	-	-	-	-	-	-	-	0	5	0	245
8	-	-	-	-	-	-	-	-	0	5	250
9	-	-	-	-	-	-	-	-	-	0	0
P	-	-	-	-	-	-	-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

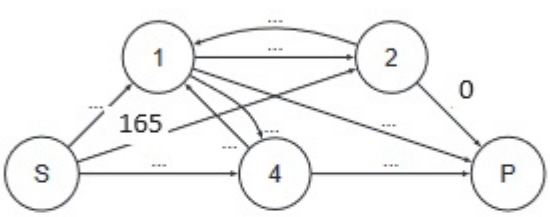
13)
Lorsque les valeurs ne sont pas écrites sur une image, cela veut dire que la valeur est la même que sur l'image précédente.



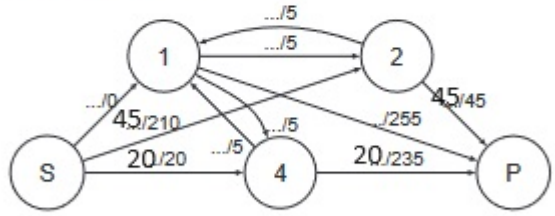


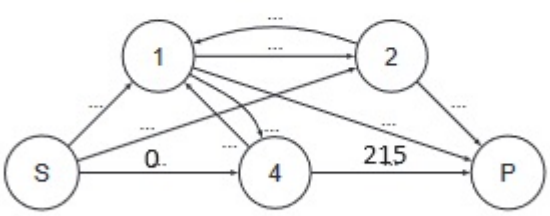
Chemin choisi : S -> 2 -> P



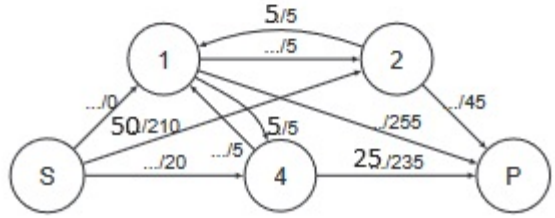


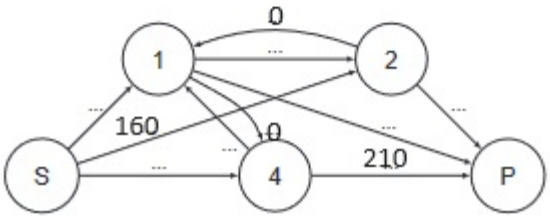
Chemin choisi : S -> 4 -> P





Chemin choisi : S -> 2 -> 1 -> 4 -> P





14)
A = {S, 2}, B = {1,4,P}.
La valeur du flot est alors 5 + 45 + 20 + 0 = 70.

15)
B correspond au pixel en noir et A au pixel en blanc sur l'image après traitement, car un pixel blanc P (valeur = 255) sera toujours accessible depuis S car la valeur totale maximale sortante de P est 5 + 5 + 5 + 5 = 20 et l'arête (S,P) a une capacité de 255.

16)
```
SELECT id FROM FONTES
WHERE nom = "Zurich"
AND style =n "romain
AND taille <= 16
AND taille >= 10
```

17)
```
SELECT fichier FROM CARACTERES
JOIN SYMBOLES ON SYMBOLES.id = CARACTERES.id_symbole
WHERE label = "A"
```

18)
```
SELECT SYMBOLES.label, count(*) FROM CARACTERES
JOIN SYMBOLES ON SYMBOLES.id = CARACTERE.id_symbole
JOIN FONTES ON FONTES.id = CARACTERE.id_fontes
WHERE nom = "Zurich"
AND style =n "romain
AND taille <= 16
AND taille >= 10
GROUP BY SYMBOLES.id
```

19)
Ce qui est obtenu par renvoie de l'appel de la fonction **lire_symbole_fichier** :
- car : ["Zurich Light BT"]
- num : 10
- var : "majuscules"
- ind : 0

La fonction renvoie "K".

20)

```
def lire_donnees_ref(fichiers_car_ref) -> dict :
    reponse = []
    for nom_fichier in fichier_car :
        nom_symbole = lire_symbole_fichier(nom_fichier)
        img(symbole) = imread(nom_fichier)
        reponse[nom_symbole] = img_symbole
    return reponse
```

21)

```
def distance(im1:array,im2:array) -> float :
    dist_c = 0
    n0, n1, _ = dimension(im1)
    for i in range(n0) :
        for j in range(n1) :
            dist_c += (im1[i][j] - im2[i][j])**2
    return dist_c ** 0.5
```

22)

```
def calcul_distance(carac_ref:dict,caract_test:array) -> dict :
    rep = {}
    for cle in carac_ref:
        rep[cle] = distance(carac_ref[cle],carac_test)
    return rep
```

23)

La question est hors programme...
Le tri fusion est un méthode de tri performante, qui est en O(nln(n)).

24)

Question aussi hors programme sans description de l'algorithme...
Le rapport admet que la question n'est pas adéquate. Il y a d'ailleurs pas mal d'erreurs dans cette question (tri par insertion classique -> deux boucles).

```
[...]
for j in range (K-1,-1,-1) :
    if voisins[j][0] > d :
    [...]
        while k > j+1 and voisins[k-1][0] > d
        [...]
    ## Erreur sujet : voisins[k] = (d,lettre)
```

25)

La complexité est en O(K * K * n). (3 boucles imbriquées avec des opérations en O(1) à l'intérieur. Si K est petit, alors O(n).
Comme un tri classique est en O(n ln(n)), c'est mieux.

26)

```
def symbole_majoritaire(voisins) :
    compte_symbole = {}
    for d,l in distances :
        if l in rep :
            rep[l] += 1
        else :
            rep[l] = 1
    max = 0
    lmax = ""
    for l in rep :
        if rep[l] > max :
            max = rep[l]
            lmax = l
    return lmax
```

27)

On peut observer ces impacts :

— Plus d'éléments dans la base de données améliore la reconnaissance
— Augmenter le nombre de fontes permet d'améliorer la reconnaissance, même en rajoutant des fontes qui ne sont pas celle utilisée
— Augmenter le nombre de voisins dans l'algorithme des K plus proche voisins n'a apparemment pas d'impact.

Augmenter K n'a pas d'impact car on essaye de reconnaître un document écrit par ordinateur. Il n'y a donc normalement pas beaucoup de bruit dû au procédé si les scans sont de bonne qualité.
Pour de la reconnaissance d'écriture humaine, il faudrait sûrement avoir un K plus élevé.